D/

WHAT HAVE WE LEARNED IN THE LAST 6 YEARS

N83 32357

MEASURING SOFTWARE DEVELOPMENT TECHNOLOGY

BY

FRANK E. McGARRY
GODDARD SPACE FLIGHT CENTER

In late 1976, the Goddard Space Flight Center (GSFC) initiated efforts to create a software laboratory where various software development technologies and methodologies could be studied, measured and enhanced. This laboratory became known as the Software Engineering Laboratory (SEL), and since its inception has been actively conducting studies and experiments utilizing flight dynamics projects in a production environment. The SEL evolved to a full partnership in the efforts between GSFC, the University of Maryland and Computer Sciences Corporation (CSC).

The approach that the SEL has taken in carrying out the studies has been to apply varying methodologies, tools, management concepts, etc. to software projects at Goddard; then to closely monitor the entire development cycle so that the entire process and product can be compared to similar projects utilizing somewhat different approaches. This monitoring function led to a need to collect, store and interpret great amounts of data pertaining to all phases of the software process, product, environment and problem. This data collection and data processing process has been applied to over 40 software projects ranging in size from 2,000 lines of code to approximately 120,000 lines of code with the typical project running about 55,000 lines of code.

The data that has been collected (and is still being collected) and interpreted for these projects comes from 5 sources:

1. Data Collection forms utilized by programmers, managers and support personnel. Typical types of data collected include:

   - o Error and Change Information
   - o Weekly Hours and Resources
   - o Component Effort (hours expended on each component by week)
   - o Project Characteristics
   - o Computer Run Analysis
   - o Change and Growth History (week by week records of source code)

(Additional Information is contained in references 1 and 2)

2. Computer Accounting Information

3. Personnel Interviews-during and after the development process

4. Management and Technical Supervisor Assessments

5. Tools-used to extract data and measures from source code

For the more than 40 projects which have been monitored, approximately 21,000 forms have been processed and are continually used to perform studies of the software development process. To support the storage, validation and usage of this information, a data base was designed and built on a PDP-11/70 at Goddard. (Reference 3)

Approach (Chart 2)

The steps that have been taken to carry out the investigation within the SEL have been:

1. Develop a profile of the software development process as it is 'now'. First we must understand what we do well and what we do not so well so we can build a baseline of current characteristics whereby later we can honestly measure change.

2. Experiment with similar type projects. The second step has been to apply select tools, methodologies and approaches to software projects so they can be studied for effect.

3. Measure the process and product. As projects are developed which are utilizing different software development techniques, the SEL uses the extracted data to determine whether or not the applied technology has made any measurable impact on the software characteristics (This may include reliability, productivity, complexity, etc.).

Environment (Chart 3)

The projects which have been monitored and studied are primarily all flight dynamics related software systems. This software includes applications to support attitude determination, attitude control, maneuver planning, orbit adjust and general mission analysis.

The attitude systems normally have very similar characteristic and all are designed to utilize graphics as well as to run in batch mode. Depending on the problem characteristics, the typical attitude systems range in size from 30,000 to over 120,000 lines of code.* The percentage of reused code ranges from less than 10 percent to nearly 70, percent with the average software package being comprised of approximately 30 percent reused code.

The applications are primarily scientific in nature with moderate reliability requirements and normally are not required to run in real time. The development period typically runs for about 2 years (from Requirements Analysis through Acceptance Testing). The development computers are typically a group of IBM S/360's which have very limited resources and where reliability is quite low (typically less than 3 hours MTBF)

Details describing the environment can be found in Reference 1.

*Here, a line of code is any 80 byte record processable by a compiler or assembler (i.e., comments are included)

Experiments Completed (Chart 4)

As was mentioned earlier, the SEL has monitored over 40 software development projects during the 6 years of operation. During this time period, numerous methologies, models, tools and general software approaches have been applied and measured. The summary results to be presented are based on these projects. The summary will be divided into 3 topic areas:

1. Profiles of the Development Process
2. Models
3. Methodologies

Profiles of the Development Process (Charts 5 thru 12)

The first step in attempting to measure the effectiveness of any software technology is to generate a baseline or profile of how one typically performs his job. Then as modified approaches are attempted on similar projects, the effects may be apparent by comparison.

Resources Allocation (Chart 7)

One set of basic information that one may want to understand is just where do programmers spend their time. When the SEL looked at numerous projects to understand where the time was spent, it found that the SEL environment deviated somewhat from the old 40-20-40 rule. Typically projects indicated that when the total hours expended were based on phase dates of a project (i.e., a specific data defined the absolute completion of one phase of the cycle and the beginning of the next phase) the breakdown was less than 25 percent for design, close to 50 percent for code and about 30 percent for integration and test.

When the programmers provided weekly data attributing their time to the activity that they felt they were actually doing, no matter what phase of software development they were in; the profile looks quite different. The 3 phases (design, code, test) each consumed approximately the same percent effort and over 25 percent of the time was attributed to 'other' activities (such as travel, training, unknown, etc.). The SEL has continually found that this effort (other) exists, and cannot easily be reduced, and most probably should be accepted as a given. The SEL has found it to be a mistake to attempt to increase productivity merely by eliminating major portions of this 'other'time.

Development Resources (Chart 8)

Another area of concern to the SEL in defining the basic profile of software development, was that of staffing level and resource expenditure profiles. Many authorities subscribe to the point that there is an optimal staffing level profile which should be followed for all software projects. Such profiles as a Rayleigh Curve are suggested as optimal. Chart 8 depicts characteristics of classes of projects monitored in the SEL and shows the difference in productivity and reliability for groups of projects having different staffing level profiles. Although the Rayleigh Curve may be acceptable for some projects, the SEL has found that wide variations on these characteristics still lead to a successful projects. The SEL has also found that extreme deviations may be indicative of problem software.

(Detailed information can be found in Reference 4 and 5)

Productivity for large vs. small systems (Chart 9)

The common belief by many software managers and developers is that as the size of a software system increases, its complexity increases at a higher rate than the lines of code increase. Because of this fact, it is commonly believed that in the effort equation

$$E = aI^b$$

where E = effort of person time
where I = lines of code

that the value of b must be greater than 1. The projects that the SEL has studied have been unable to verify this belief and instead have found the value of b to approximate .92 in the SEL environment. The fact that this equation is nearly linear leads to the counter intuitive point that a project of 150,000 lines of code will cost approximately 3 times as much as a 50,000 lines of code project-instead of 4 or 5 times as much as is often commonly believed.

(Further details can be found in Reference 6.)

Productivity Variation (Chart 10)

Another characteristics that the SEL has been interested in studying has been the variations in programmer productivity. Obviously one would want to increase the productivity by whatever approach found to be effective, but first we must clearly understand what the baseline characteristics of productivity are (minimum, maximum, average, difference between small and large projects, etc.); only then will we know if we have improved or not in the years to come.

As has been found by other researchers in varying environments, the productivity of different programmers can easily differ by a factor of 5 or 10 to 1. The SEL did find that there was a greater variation (from very low productivity of .5 l.o.c/hour to 10.8 l.o.c./hour) in small projects. The probable reason for this is that newer people are typically put on smaller projects and the SEL has found extreme differences in the relatively inexperienced personnel.

Reusing Code (Chart 11)

As was stated in the introduction, projects being developed in the SEL environment typically utilize approximately 30 percent old code. Although it is obviously less costly to integrate existing code into a system rather than having to generate new code, there is some cost that must be attributed to adopting the old code. The development team must test, integrate and possibly document the old code, so there is some overhead. By looking at approximately 25 projects ranging in size from 25,000 lines of code to over 100,000 total lines of code and ranging in percent of reused code from 0 percent to 70 percent, the SEL finds that by attributing a value of approximately 20 percent overhead cost to reuse code, the expenditures of the 25 projects can best be characterized. Now the SEL uses the 20 percent figure for estimating the cost of adopting existing code to a new software project.

Error Characteristics (Chart 12)

One of the other characteristics of a software environment that is of great concern to developers and managers is that of expected software reliability and that of overall software error characteristics. Before attempting to improve software reliability or before attempting to minimize the impact that software errors may have, the SEL had to first understand the error characteristics of the typical applications software in the SEL environment.

By collecting detailed error report data and through the monitoring of numerous applications projects many error characteristics have been studied.

Several pieces of information which are depicted in Chart 12 and which are based on 1381 error reports from approximately 15 projects include:

- Most errors are local to one component (subroutine or function)
- Less than 10 percent of errors were attributed to faulty requirements
- A great percent of errors (48 percent) were estimated to be trivial to correct (less than 1 hour)
- A very low percent of errors (7 percent) were estimated to be a major effort to fix (greater than 3 days)

(Further statistics and more detailed explanations can be found in References 7 and 8).

Models (Charts 11 through 16)

A second set of studies that the SEL has actively pursued is that of evaluating, reviewing, and developing software models. This includes resource models, reliability models as well as complexity metrics.

Measures for Software (Chart 14)

The SEL has attempted to utilize various available software metrics to characterize the software products generated. Such metrics as the McCabe Cyclomatic Complexity, Halstead Length, and lines of code were only a few of the measures that were reviewed.

It is commonly believed that the size of a component or the complexity of a component will be directly correlated to the reliability of that component. One set of studies performed in the SEL attempted to verify this belief. By taking over 650 modules which had very detailed records of error data, the SEL compared the correlations of 4 characteristics of the components. The characteristic included total lines of code, executable lines of code, Cyclomatic Complexity and Halstead Length. The resultant correlations are dipicted in Chart 14; which shows a very high direct correlation for the 4 measures.

A second study was performed where the error rate of each of the components was plotted against size as well as against Cyclomatic Complexity. The SEL expected to show that larger components have higher error rates than smaller components and that components of higher complexity rating had higher error rates. The plots on Chart 11 show that the results were counter-intuitive. The SEL has been unable to verify that larger or more complex components indeed have higher error rates.

Cost Models (Chart 15)

In addition to the studies made pertaining to various measures for software, the SEL has also utilized the cost data collected from the many projects to calibrate and evaluate various available resource estimation models. No attempt was intended to qualify one model as being any better than another. The objective of the studies was to better understand the sensitivities of the various models and to determine which models seemed to characterize the SEL software development environment most consistently.

In studying these resource models, 9 projects which were somewhat similar in size were used as experimental projects. Each of the models was fed complete and accurate data from the SEL data base and each was calibrated with nominal sets of projects as completely as the experimenters could. Summary results, which are given in Chart 15, indicate that, occasionally, some models can accurately predict effort required for a software project. The SEL has

reiterated what many other software developers and managers claim. Cost models should never be used as a sole source of estimation. The user must have access to experienced personnel for estimating and
must also have access to a corporate memory which can be used to calibrate and reinforce someones estimate of cost. Resource models can only be used as a supplemental tool to reinforce ones estimate or to flag possible inconsistencies.

More detailed information on the SEL studies can be found in Reference 1, 9, 10, 5

Reliability Models (Chart 16)

Another type of model that the SEL has spent some efforts in understanding and calibrating is the reliability model. Although numerous approaches have been suggested as to just how one best predicts the level of error proneness that software may have, the SEL has only performed any extended studies on one model-that which is attributed to John Musa. The model is a maximum likelihood method and the SEL attempted to apply detailed fault reports from 2 separate projects to the model in an attempt to determine if the model could accurately predict remaining faults in the software.

Chart 16 indicates that one of the experiments was quite successful and one of the experiments was not successful. It should be noted that during and after these experiments, John Musa reviewed the results and the data very carefully and he has pointed out some possible deficiencies in the SEL data which could possibly lead to erroneous results in this application of the reliability model. One such piece of data is the granularity with which computer CPU time is recorded between reported faults. The SEL data is not as accurate as the model calls for.

The charts show that for experiment 1, the model quite accurately predicted a level of reliability after approximately 1/2 of the total uncovered faults were reported. The chart also shows that for experiment 2, the model was still predicting a very high number of errors to be still in the software, when in fact a minimal set were ever uncovered during the several years of operation for that system.

More detailed discussions can be found in Reference 1 and 11.

Methodologies (Charts 17 through 20)

As was mentioned earlier, one of the major objectives of the SEL has been to measure the effectiveness of various software development methodologies. The SEL has utilized selected development approaches in different applications software tasks and then has analyzed the process and product to study the relative impact of the approach. A summary of some of the results of the experimentation process is presented here.

Use of An Independent Verification and Validation Team (Chart 18)

Many software managers, developers and organizations have advocated the usage of an independent IV&V team during the software development process. The major advantage of following such an approach, it is claimed, will be the improvement in software reliability, quality, visibility, but not necessarily an improvement in overall software productivity.

In an attempt to evaluate the impact that the usage of an IV&V team may have on the SEL environment, 3 candidate projects were selected to utilize the methodology of an IV&V. Two of the projects were very typical flight dynamics systems, each containing over 50,000 lines of code while the third was a smaller flight dynamics project comprised of about 10,000 lines of code. In addition to the IV&V approach being applied to the projects, the development teams utilized the commonly followed standards and approaches normally used by development efforts within the SEL environment.

The projects lasted approximately 18 months, and the IV&V effort was active for the entire duration of the project. The size of the IV&V effort was about 18 percent of the effort of each of the large development efforts. A series of measures was defined near the beginning of the experiment by the SEL. These measures would be used to determine whether or not the application of the IV&V approach was cost effective in the SEL environment.

A summary of some of the measures is depicted in Chart 18. The results here indicate:

- o total cost of the project increased-as expected
- o productivity of the development teams (not counting the cost of IV&V) was among the lowest of any previous SEL monitored project.
- o rates of uncovering errors found earlier in the development cycle was better
- o cost rate to fix all discovered errors was no less than in any other SEL projects
- o reliability of the software (error rate during acceptance testing and during maintenance and operations) was no different than other SEL projects

The conclusion of the SEL, based on these 3 experiments, was that the IV&V methodology was not an effective approach in this SEL environment.

(A more detailed description can be found in Reference 12).

Effects of MPP on Software Development (Chart 19)

In an attempt to determine if the utilization of Modern Programming Practices (MPP) has any impact (either favorable or unfavorable) on the development of software, a set of 10 fairly large (between 50,000 l.o.c. and 120,000 l.o.c.), and fairly similar projects (same development environment, same type of requirements, same time constraints) was closely examined. These projects all had been developed in the SEL environment where detailed information was extracted from the projects weekly and where each project had a different level of MPP enforced during the development process.

The MPP's ranged from various design approaches (such as PDL, Design Walk Throughs, etc.) to code and test methodologies (such as structured code, code reading, etc.), to various integration and system testing approaches. All of the possible MPP's were rated and scaled as to the level to which the practice was followed for each project (the rating was done by the SEL researchers, not by the software developers). The only purpose of this exercise was to depict trends and not to prove that any one single practice was more effective by itself than any other.

The level to which MPP's were utilized were plotted against productivity and against error rate. Chart 19 indicates that the application of the MPP has favorably affected productivity by about 15 percent for these experiments. The results of software reliability vs MPP is very questionable. The SEL is still continuing analysis of additional data. The chart shown is obviously very inconclusive.

(More details of this effort can be found in Reference 13).

Subjective Summary of Effective Practices (Chart 20)

The previous chart indicated that productivity can be improved by an appreciable amount if certain, select practices are applied to the software development process. One obviously next would ask, which practices are the most effective? The SEL has been attempting to analyze the available data from the 40 experiments it has conducted to answer this very question. As was stated earlier, the SEL feels that these types of experiments can only depict trends and cannot accurately isolate one practice as measurable on its own. Whether or not this can be done, or whether one should ever attempt it is questionable. Most software development methodologies represent an integrated set of practices that only are effective when they are applied in a combined, uniform fashion. Most practices do not make sense, or at least cannot be effective as a stand alone approach.

A summary of the trends that the SEL has discovered for specific experiments conducted is represented in Chart 20. This chart is a combination of experimental results and subjective information from the experimenters and users and should only be viewed as depicting trends in various approaches. No numerical value of impact can realistically be assigned to the individual practices tested. It seems that practices such as PDL, code reading and librarian have proved most beneficial while such techniques as automated flow charters, requirements languages and the axriomatic design approach have been unsuccessful in the SEL.

Cost of Data Collection (Chart 21)

The SEL has been in existence for about 7 years and has been collecting detailed software development data for over 6 years. Numerous experiments have been conducted in an attempt to understand and measure various methodologies for developing software. In support of these efforts, one of the most critical and difficult elements of the entire experimentation process is that of data collection.

The data collection process is time consuming, frustrating, sometimes unrewarding, and most assurably is expensive. Chart 21 shows the overhead cost that the SEL has experienced over the past 6 years. To accurately collect data from the development tasks, the SEL finds that there is a 3 to 7 percent overhead price on the development effort. To process the data that has been collected (verification, encoding, data entry, storage, etc.), the SEL has spent approximately an additional 10 to 12 percent of the development effort. Finally, the SEL experiences indicate that one can spend up to an additional 25 percent of the development effort to perform the detailed analysis of the data that has been collected. This includes support before, during and after the experiments in defining the data to be collected, monitoring the development data and effort, formulating hypothesis and performing analysis of the completed experiments. The product of the analysis consists of papers, reports, and documents.

(Detailed information on cost can be found in Reference 2).

Summary (Chart 22)

In summary, the SEL has had much experience with the data collection process and with the experimentation process. Many of its attempts have been rewarding and many have been fruitless, but the SEL feels attempts to assess approaches to software have to be conducted if we are ever to evolve to a more productive approach to developing software.

REFERENCES

1. Software Engineering Laboratory, SEL 81-104, The Software Engineering Laboratory, D.N. Card, F. E. McGarry, G. Page, et. al., February 1982

2. SEL, 81-101, Guide to Data Collection, V. E. Church, D. N. Card, F. E. McGarry, et. al., August 1982

3. SEL, 81-102, Software Engineering Laboratory (SEL) Data Base Organization and User's Guide, D. C. Wyckoff, G. Page, F. E. McGarry, et. al., March 1983

4. Zelkowitz, M. V., "Resource Estimation for Medium Scale Software Projects", Proceedings of the Twelfth Conference on the Interface of Statistics and Computer Science, New York, Computer Societies Press, 1979

5. Bailey, J. W., and V. R. Basili, "A Meta-Model for Software Development Resource Expenditures", 'Proceedings of the Fifth International Conference on Software Engineering', New York; Computer Societies Press, 1981

6. Basili, V. R., and K. Freburger, 'Programming Measurement and Estimation in the Software Engineering Laboratory', Journal of Systems and Software, February 1981, Volume 2, No. 1

7. SEL 81-011, Evaluating Software Development by Analysis of Change Data, D. M. Weiss, November 1981

8. Basili, V. R., and B. T. Perricone, Software Errors and Complexity: An Empirical Investigation, University of Maryland, Technical Report TR-1195, August 1982

9. SEL 80-007, An Appraisel of Selected Cost/Resource Estimation Models for Software Systems, J. F. Cook, F. E. McGarry, December 1980

10. Basili, V. R., 'Software Engineering Laboratory Relationships for Programming Measurment and Estimation', University of Maryland, Technical Memorandum, October 1979

11. SEL 80-005, A Study of the Musa Reliability Model, A. M. Miller, November 1980

12. SEL 81-110, Performance and Evaluation of an Independent Software Verification and Integration Process, G. Page. and F. McGarry, September 1982

13. SEL 82-001, Evaluation of Management Measures of Software Development, D. Card, G. Page, F. McGarry, September 1982

MEASURING SOFTWARE DEVELOPMENT TECHNOLOGY

OR

SHOULD PROGRAMMERS DO IT TOP DOWN ?

F. McGarry
NASA/GSFC

CHART 1

SEL APPROACH TO SOFTWARE TECHNOLOGY ASSESSMENT

SOFTWARE EXPERIMENTS IN PRODUCTION ENVIRONMENT: NASA APPLICATIONS

- **DEVELOP PROFILE OF ENVIRONMENT (SCREENING)**
  - EXTRACT DETAILED DEVELOPMENT DATA
  - DETERMINE CHARACTERISTICS OF DEVELOPMENT PROCESS
- **EXPERIMENT WITH PROPOSED TECHNOLOGIES (CONTROLLED)**
  - APPLY VARIOUS TECHNOLOGIES (METHODS, MODELS, AND TOOLS) TO APPLICATIONS PROGRAMS
  - EXTRACT DETAILED DEVELOPMENT DATA
- **MEASURE IMPACT AND/OR ASSESS TECHNOLOGIES**
  - DEFINE MEASURES FOR EVALUATION
  - COMPARE EFFECTS OF USING OR NOT USING APPROACHES IN QUESTION (SIMILAR PROJECTS)
  - DETERMINE EFFECTIVENESS OF TECHNOLOGIES IN QUESTION (WHICH ONES HELP AND BY HOW MUCH)

F. McGarry
NASA/GSFC

SOFTWARE ENVIRONMENT

DEVELOPMENT LANGUAGE	FORTRAN (15% MACROS)
SOFTWARE TYPE	SCIENTIFIC, GROUND-BASED INTERACTIVE, NEAR-REAL-TIME
SIZE	TYPICALLY~60,000 SLOC (2,000 TO 110,000)
DEVELOPMENT TIME	16 TO 24 MONTHS (START DESIGN TO START OPERATIONS)
STAFFING	6 TO 14 PERSONS
DEVELOPMENT SYSTEM	IBM S/360 (PRIMARILY) VAX-11/780 PDP-11/70

EXPERIMENTS WITHIN THE SEL 1977 THROUGH 1982 BASIS FOR SUMMARY INFORMATION AND CONCLUSIONS

LABORATORY EXPERIMENTS	46 PROJECTS
INFORMATION MONITORED	1.8 MILLION SLOC
PROGRAMMERS/MANAGERS REPRESENTED	150 PEOPLE
DATA EXTRACTED	20,000 FORMS
METHODOLOGIES APPLIED	200 QUALIFYING PARAMETERS AND VARIOUS MODELS, TOOLS, AND METHODOLOGIES

334-PAG-(2*)

AREAS OF DISCUSSION

- PROFILES
- MODELS
- METHODOLOGIES

334-PAG-(2*)

F. McGarry
NASA/GSFC

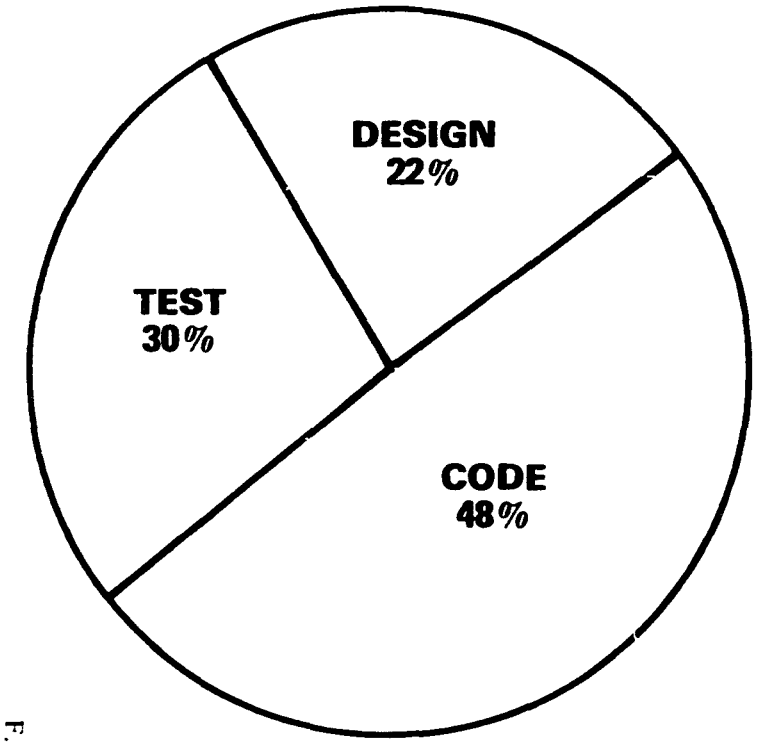
CHART 5

PROFILES

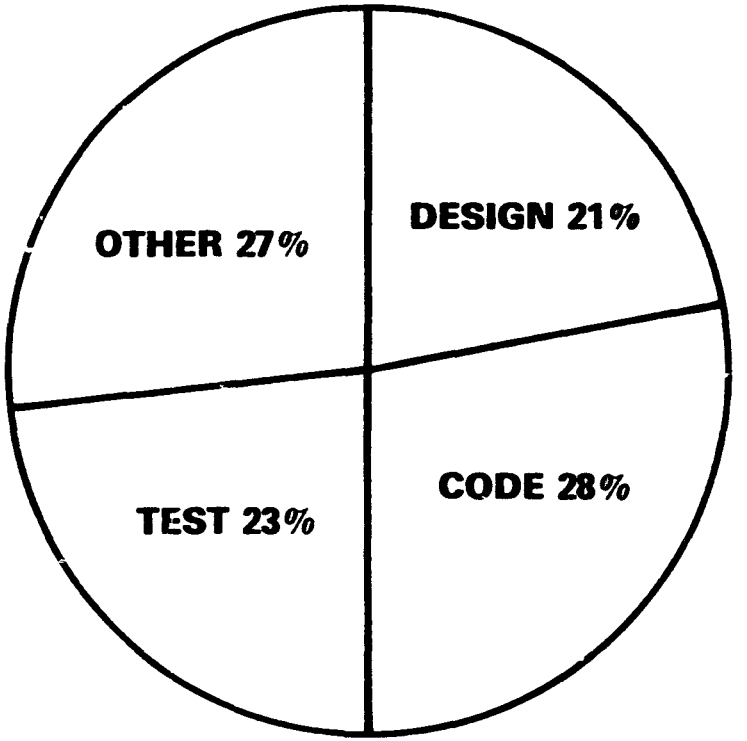
334-PAG-(2*)

WHERE DO PROGRAMMERS SPEND THEIR TIME?

DATE DEPENDENT



PROGRAMMER REPORTING

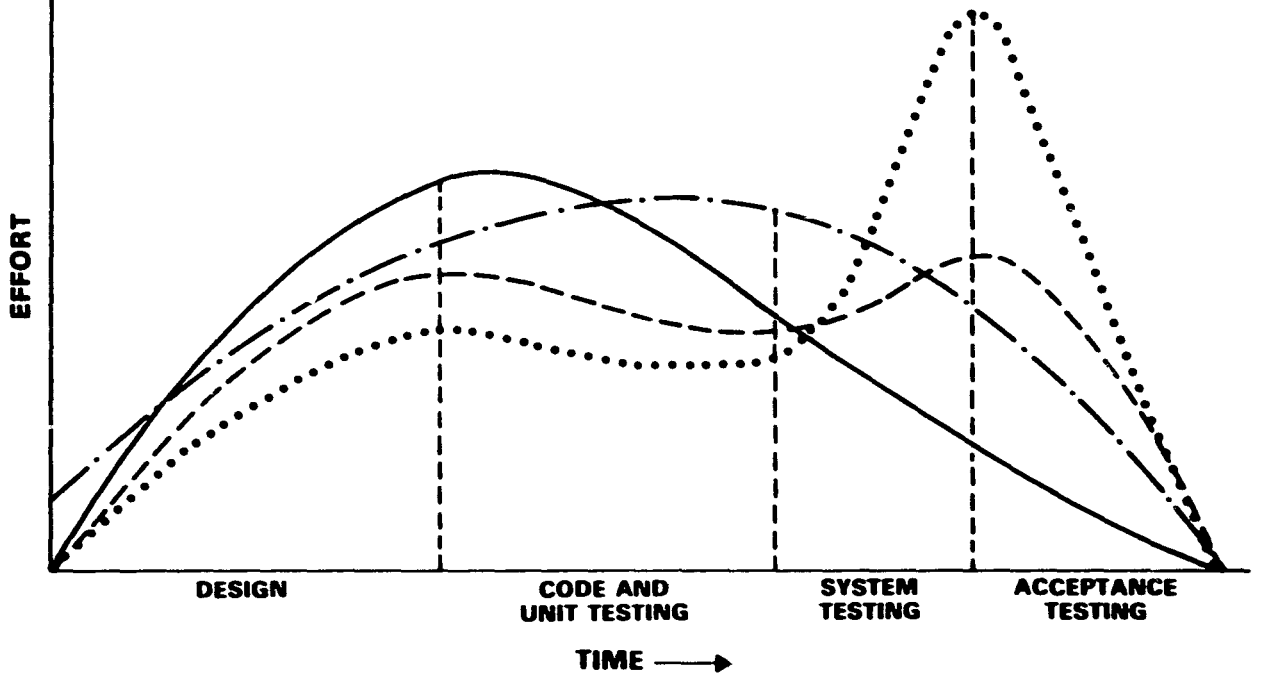


334-PAG-(2*)

F. McGarry
NASA/GSFC

CHART 7

PROFILES OF DEVELOPMENT RESOURCES
HOW MANY ROADS TO COMPLETION?



PROFILE	PRODUCTIVITY (SLOC/HOUR)	RELIABILITY (ERRORS/K SLOC)
———	RAYLEIGH CURVE	–
—·—	4.4–4.6	UP TO 2
– – –	2.7–4.7	UP TO 2
•••••••	2.7–2.9	UP TO 2

- RELATIONSHIP BETWEEN PROFILE AND PRODUCTIVITY
- NO RELATIONSHIP BETWEEN PROFILE AND RELIABILITY

CHART 8

ARE LARGE PROGRAMS HARDER TO BUILD THAN SMALL ONES?

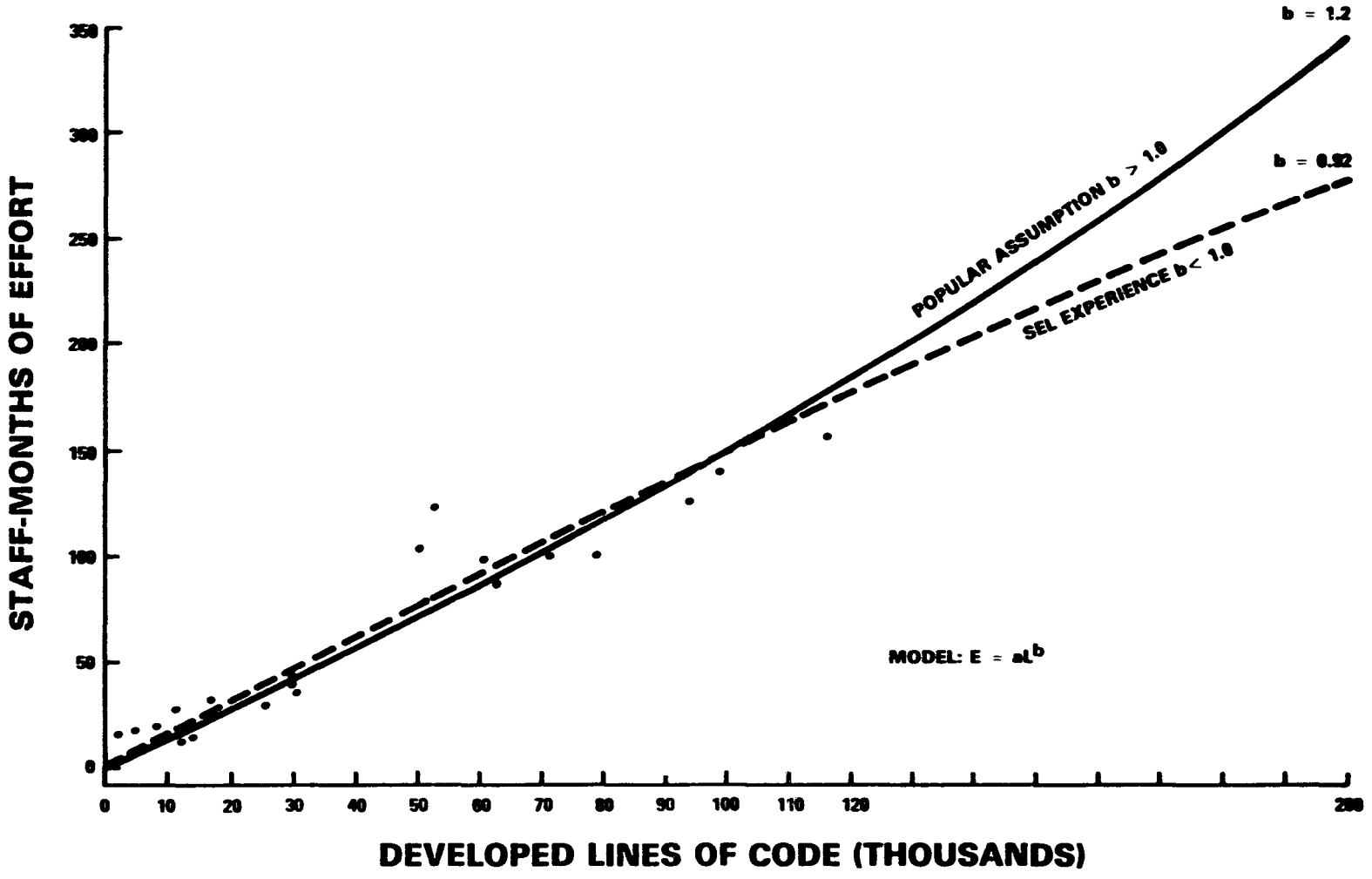
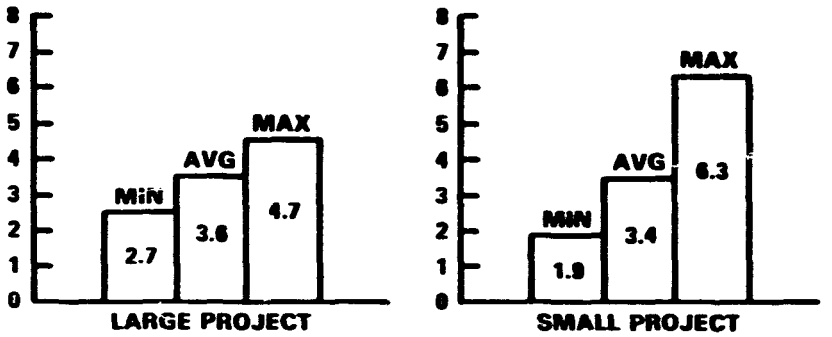


CHART 9

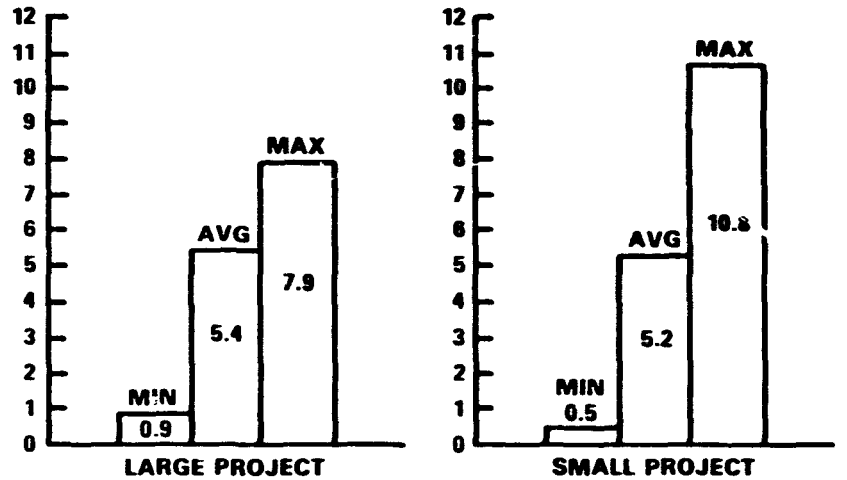
334-PAG-(2c*)

PRODUCTIVITY VARIATION (SLOC/HOUR)[1]

BY PROJECT (ALL CHARGES)



BY PERSON (PROGRAMMER ONLY)



PEOPLE ARE THE MOST IMPORTANT METHODOLOGY

[1] A LARGE PROJECT IS GREATER THAN 20K SLOC.

224-PAG-(2a*1)

ASSESSING REUSED CODE

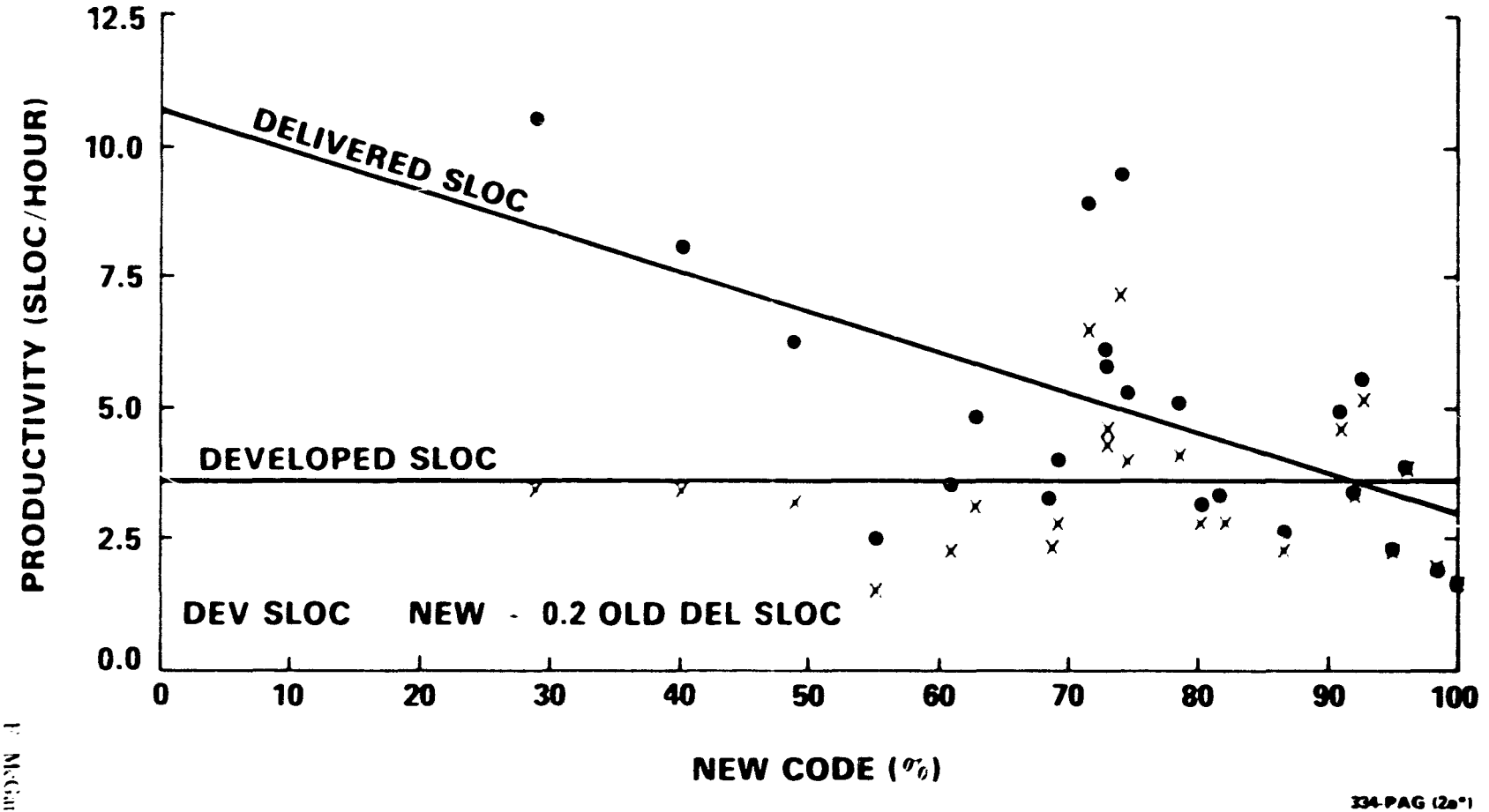
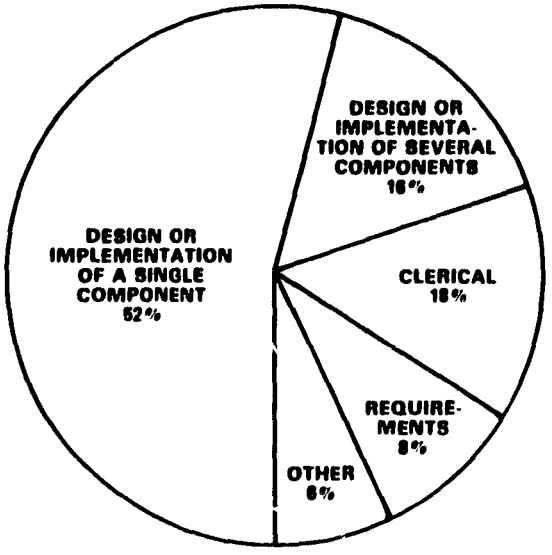


CHART 11

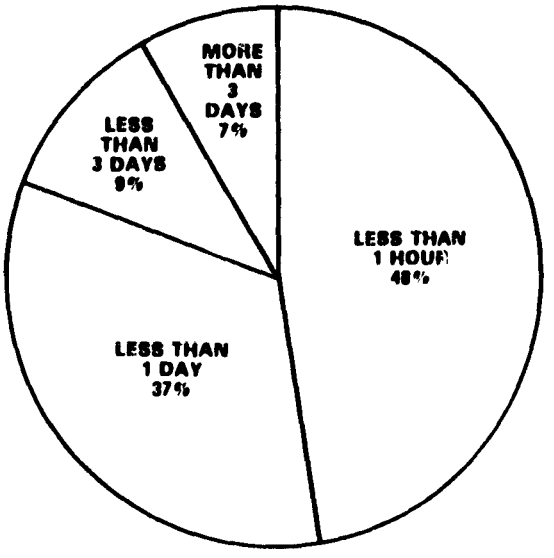
ERROR CHARACTERISTICS
(MEASURED DURING IMPLEMENTATION)

TYPES OF ERRORS



SAMPLE OF 1381 REPORTS

EFFORT TO CORRECT



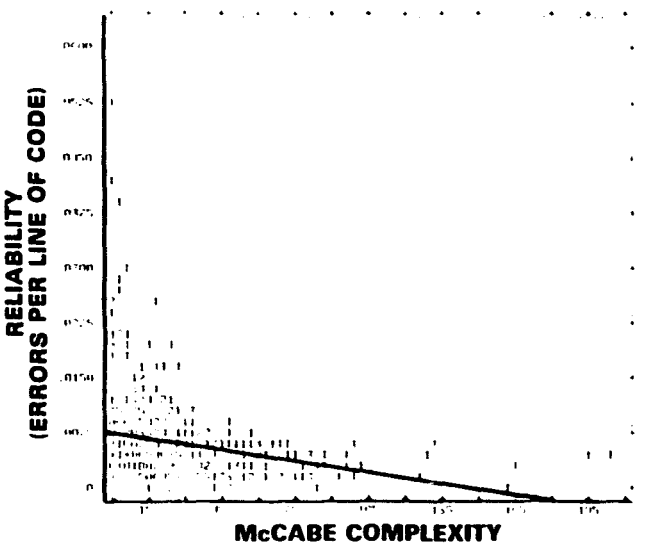
- **MOST ERRORS ARE EASY TO CORRECT**
- **SEVERAL-COMPONENT ERRORS ARE LESS THAN EXPECTED**
- **REQUIREMENTS ERRORS ARE LESS THAN EXPECTED**

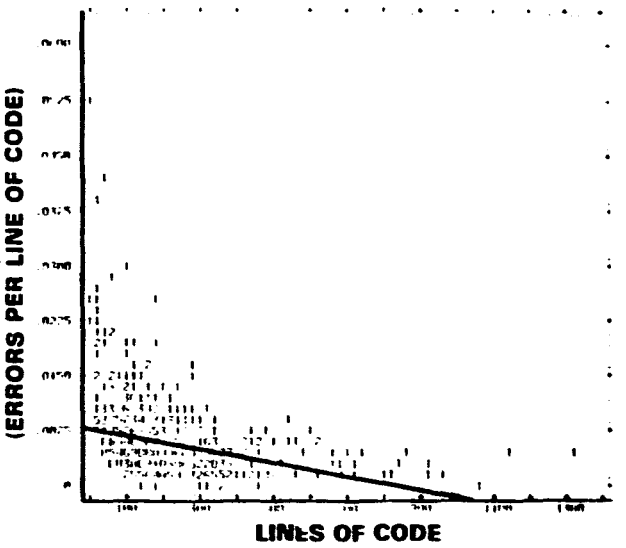
MODELS

334-PAG-(Zb*)

CHART 13

SOFTWARE MEASURES IN THE SEL





	CORRELATIONS			
	TOTAL LINES	EXECUTABLE LINES	McCAPE COMPLEXITY	HALSTEAD LENGTH
HALSTEAD LENGTH	0.85	0.91	0.91	1.00
McCABE COMPLEXITY	0.81	0.87	1.00	
EXECUTABLE LINES	0.84	1.00		
TOTAL LINES	1.00			

SAMPLE OF 688 MODULES

334-PAG-(2c*)

ORIGINAL PAGE IS
OF POOR QUALITY

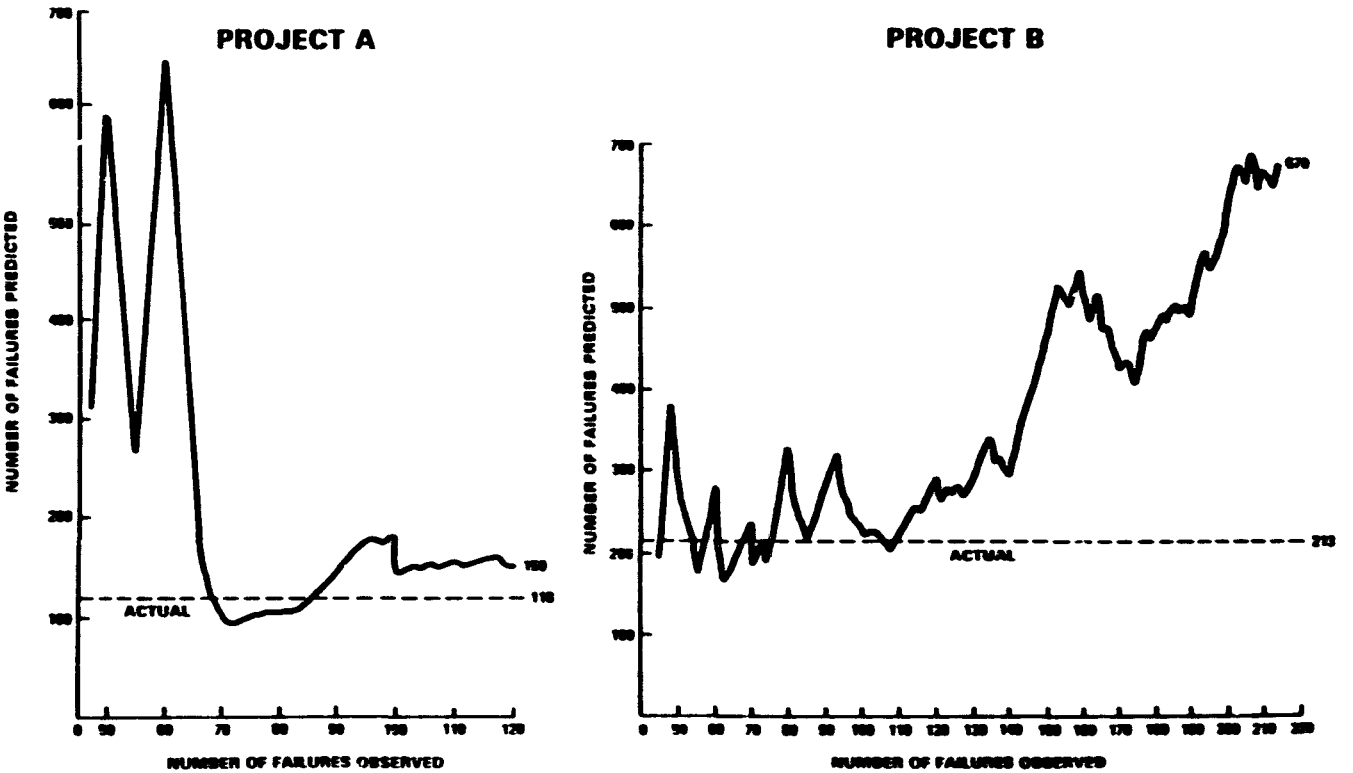
COMPARISON OF COST MODELS

PROJECT	ACTUAL EFFORT (MM)	PERCENTAGE OF ERROR IN PREDICTION				
		DOTY	PRICE S3	TECOLOTE	SEL	COCOMO
1	79	+65	+8	−4	−6	—
2	96	+30	+6	−25	−22	+1
3	40	+65	+6	−8	+93	—
5	98	+74	0	+3	−2	+2
6	116	+123	+36	+35	−3	—
7	91	+52	+14	−12	−14	—
8	99	+127	+7	+36	+14	+53
9	106	—	—	—	−24	+16

SOMETIMES, SOME MODELS WORK WELL

334-PAG-(2b*)

PREDICTING RELIABILITY
(MUSA MAXIMUM LIKELIHOOD METHOD)



WE DON'T KNOW ENOUGH ABOUT RELIABILITY MODELS

CHART 16

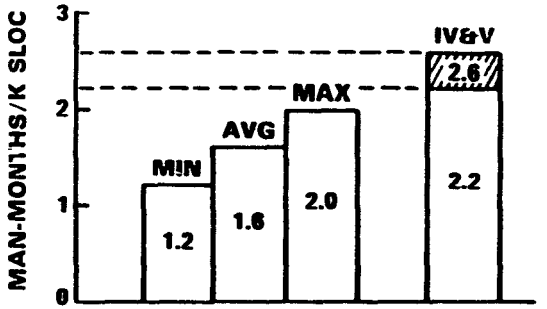
METHODOLOGIES

F. McGarry
NASA/GSFC

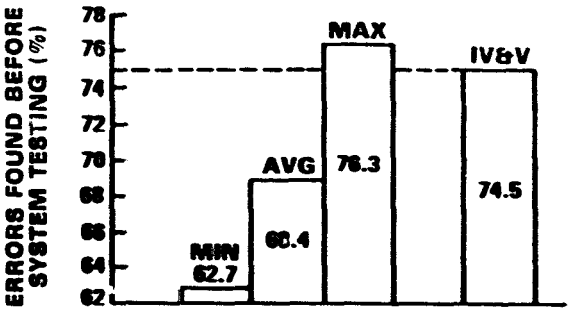
CHART 17

A LOOK AT IV&V METHODOLOGY

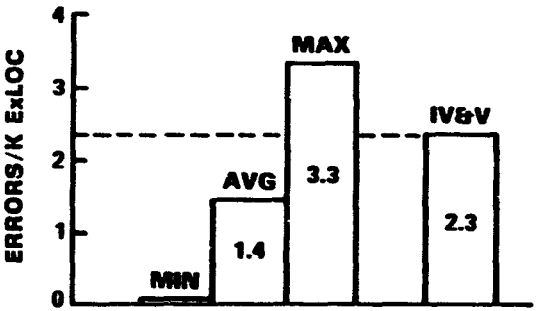
(BASED ON RESULTS FROM 3 EXPERIMENTS)



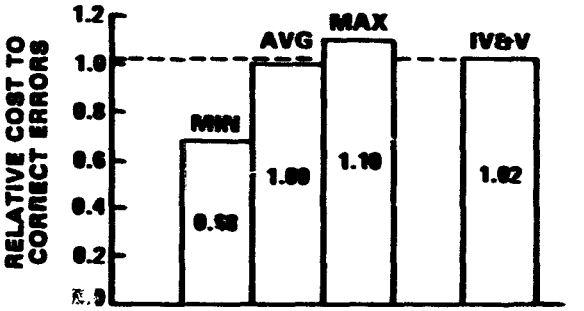
- COST INCREASED



- MORE ERRORS FOUND EARLY



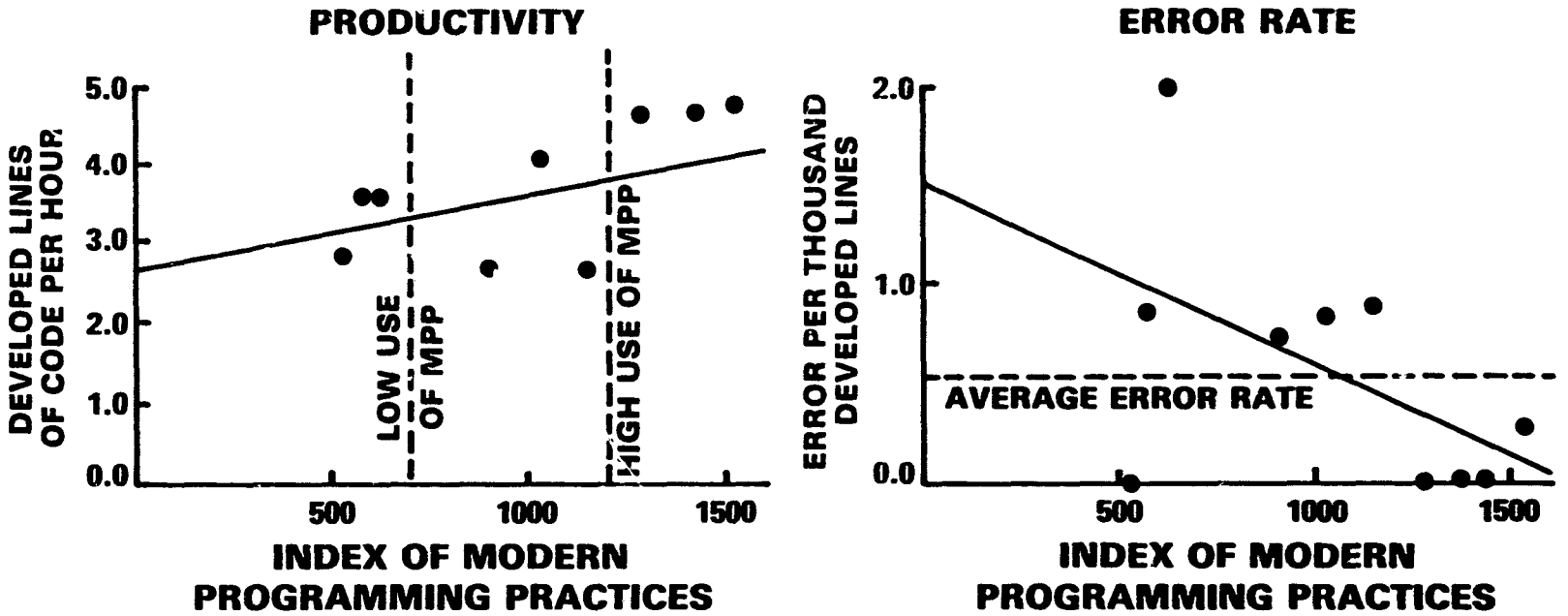
- RELIABILITY NOT IMPROVED



- ERROR CORRECTION COST NOT DIFFERENT

- **IF YOU MULTIPLY ERRORS FOUND EARLY BY A LATENCY FACTOR, IV&V LOOKS GOOD**
- **IF YOU EXAMINE ALL MEASURES, IV&V LOOKS BAD**

EFFECTS OF MPP ON SEL SOFTWARE DEVELOPMENT



- PRODUCTIVITY IS ABOUT 15 PERCENT HIGHER
- RELIABILITY IS HIGHLY VARIABLE

334-PAG-(2d*)

F. McGarry
NASA/GSFC

WHAT HAS BEEN SUCCESSFUL IN OUR ENVIRONMENT?

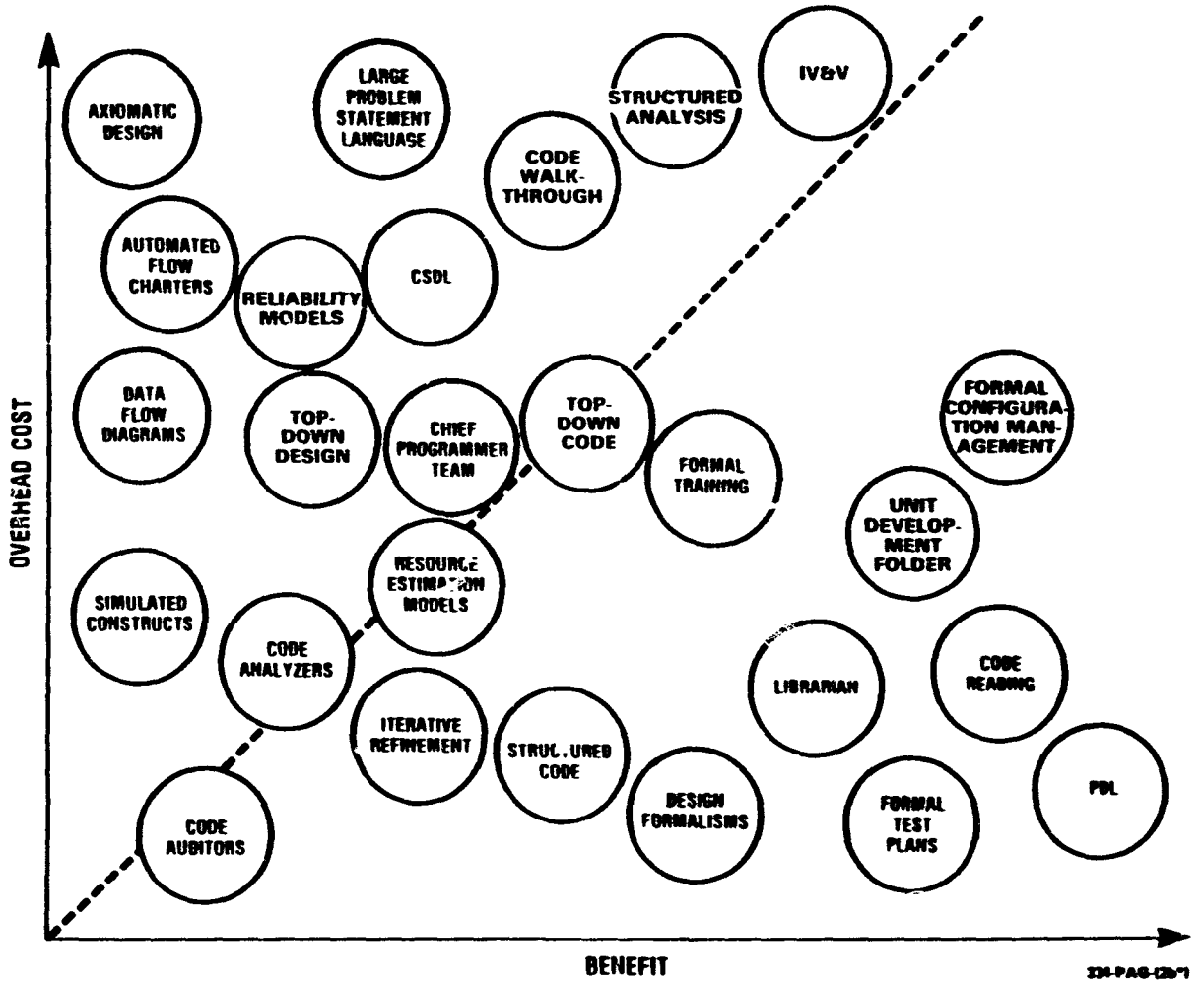


CHART 20

COST OF DATA COLLECTION
(AS A PERCENTAGE OF TASKS BEING MEASURED)

	SEL EXPERIENCES
OVERHEAD TO TASKS (EXPERIMENTS)	**3—7%**
● FORMS	
● MEETINGS	
● TRAINING	
● INTERVIEWS	
● COST OF USING TOOLS	
DATA PROCESSING	**10—12%**
● COLLECTING/VALIDATING FORMS	
● ARCHIVING/ENTERING DATA	
● DATA MANAGEMENT AND REPORTING	
ANALYSIS OF INFORMATION	**UP TO 25%**
● DESIGNING EXPERIMENTS	
● EVALUATING EXPERIMENTS	
● DEFINING ANALYSIS TOOLS	

SUMMARY

- DATA COLLECTION IS EXPENSIVE — BUT VERY, VERY IMPORTANT
- WE MUST UNDERSTAND WHERE WE ARE BEFORE HEADING SOMEWHERE ELSE
- EXPERIMENTATION WILL PAY FOR ITSELF (TRY SOMETHING NEW)
- MPP CAN FAVORABLY IMPACT PRODUCTIVITY AND RELIABILITY
- SOME METHODOLOGIES BUY YOU NOTHING (OR EVEN WORSE)
- MODELS MUST BE UTILIZED WITH GREAT CARE

CHART 22