N86-30359

STUDIES AND EXPERIMENTS IN THE* SOFTWARE ENGINEERING LAB (SEL)

BY
FRANK E. MCGARRY
NASA/GSFC
AND
DAVID N. CARD
COMPUTER SCIENCES CORPORATION (CSC)

ABSTRACT

The Software Engineering Laboratory (SEL) is an organization created nearly 10 years ago for the purpose of identifying, measuring and applying quality software engineering techniques in a production environment (Reference 1). The members of the SEL include NASA/GSFC (the sponsor and organizer), University of Maryland, and Computer Sciences Corporation. Since its inception the SEL has conducted numerous experiments, and has evaluated a wide range of software technologies. This paper describes several of the more recent experiments as well as some of the general conclusions to which the SEL has arrived.

1.0 Background (Chart 1)

Over the past 9 years, the SEL has conducted studies in 4 major areas of software technology:

1. Software Tools and Environments
2. Development Methods
3. Measures and Profiles
4. Software Models

Most of these studies have been conducted by utilizing specific approaches, tools or models to production software problems within the flight dynamics environment at Goddard. By extracting detailed information pertaining to the problem, environment, process and product, the SEL has been able to gain some insight into the relative impact that the various technologies may have on the quality of the software being developed.

More detailed descriptions of the overall measurement process as well as the SEL studies may be found in References 1, 2, and 3. This brief paper will describe some of the more recent, specific experiments that have been conducted by/in the SEL and just what types of insight may be provided in areas of:

1. Tools and Environments
2. Software Testing
3. Design Measures
4. General Trends

*The work described in this paper has been extracted from reports and studies carried out by members of the SEL.

TYPE OF SOFTWARE: SCIENTIFIC, GROUND-BASED, INTERACTIVE GRAPHIC, MODERATE RELIABILITY AND RESPONSE REQUIREMENTS

LANGUAGES: 85% FORTRAN, 15% ASSEMBLER MACROS

COMPUTERS: IBM MAINFRAMES, BATCH WITH TSO

PROJECT CHARACTERISTICS:	AVERAGE	HIGH	LOW
DURATION (MONTHS)	16	21	13
EFFORT (STAFF-YEARS)	8	24	2
SIZE (1000 LOC)			
DEVELOPED	57	142	22
DELIVERED	62	159	33
STAFF (FULL-TIME EQUIVALENT)			
AVERAGE	5	11	2
PEAK	10	24	4
INDIVUALS	14	29	7
APPLICATION EXPERIENCE (YEARS)			
MANAGERS	6	7	5
TECHNICAL STAFF	4	5	3
OVERALL EXPERIENCE (YEARS)			
MANAGERS	10	14	8
TECHNICAL STAFF	9	11	7

FIGURE 1. FLIGHT DYNAMICS SOFTWARE

The Flight Dynamics environment typically is a FORTRAN environment building software systems ranging in size from 10,000 to 150,000 lines of code - (see Figure 1).

2.0 Software Tools/Environments* (Chart 2 and Reference 4)

One of the more interesting studies that was conducted within the past several years, was one in which an attempt was made to measure the impact of several development approaches (related to environment support) on the quality of software within the flight dynamics discipline.

The three points of study include:

1. Software Tools
2. Computer Support
3. Number of Terminals/Programmer

The quality of the product was measured using 4 attributes including:

1. Productivity - Number of developed lines of code per man month.

2. Reliability - Number of errors reported per 1,000 lines of code.

3. Effort to Change - (Average number of man hours required to make a software modification).

4. Effort to Repair (Average number of man hours required to correct an identified error)

2.1 Experiment Description (Chart 3)

In carrying out the study, a review of all projects for which detailed project history data was available and complete was undertaken. From the completed 50 projects, 14 were selected because of the quality and completeness of the relevant data and more importantly because of the general similarity of complexity of problems that the software was attempting to solve.

Fourteen projects ranging in size from 11,000 lines of code to 136,000 lines of code were selected. These projects had information describing the environment under which they were developed and additional information such as the number and quality of automated tools utilized and the number of interactive terminals available to the programming staff.

*Lead investigators of this work included F. McGarry and J. Valett of NASA/GSFC and D. Hall of NASA/HQ.

The 14 projects selected all dealt with tasks in solving attitude determination and control related problems. The projects were completed between the years 1978 to 1984.

The projects also had detailed information as to manhours, size, error history, and effort required to make all changes and corrections to the software.

2.2 Project Variations (Chart 4)

In attempting to characterize each of the development projects, a ranking scheme was used for this particular study. It was found that the availability of terminals ranged from a low of less than 1 per 8 programmers to a high of better than 1 per 2 programmers.

There were a total of 21 tools considered in this study that were applied by at least some of the projects studied. Such tools as documentation aids, preprocessors, test generators and program optimizers were among the tools considered.

It was also found that the distribution of level of use for tools ranged from a low of only 1 or 2 automated tools being used, to a high of more than 8 automated tools being used. These tools also were rated as far as the actual usage by the particular project and also there was a rating for each tool of the assessed 'quality' of the particular tool. Quality here was rated for each tool on a scale of 1 to 5 and was a subjective rating determined by the software manager.

There were a total of 11 characteristics that made up the computer support measure. These 11 included:

- Terminal Accessibility
- Turn around time
- Compiler Speed
- System Reliability (2 measures)
- Direct Storage
- Offline Storage
- Interactive Availability
- Terminals/programmers
- Avg. CPU Utilization
- Accessibility of all resources

2.3 Study Results (Chart 5)

The results of this particular study were encouraging on the one hand and quite perplexing on the other.

2.3.1 Tool usage results showed that as the number and quality

of automated tools increased, there were significant increases in 3 of the 4 quality measures used in this study:

1. Productivity increased as tool usage increased

2. Maintainability (effort to change/effort to repair) improved as the number and quality of tools increased.

3. Reliability did not seem to be significantly impacted in this one particular study.

2.3.2 Computer Environment

Although all of the experimenters felt that there would be significant increases in all quality measures as the overall quality of computer support increased, none of the measures proved to be significant for this one particular study. It could not be shown that an improved computer support environment (at leastas far as the way the SEL described support environment) directly, favorably impacted the four quality measures used by the SEL.

This particular study is still undergoing further analysis.

2.3.3 Terminal Usage

The most perplexing result of this experiment study was the one in which the SEL attempted to assess the impact that increased number of terminals would have on the four measures described.

Although the experimenters expected to observe an increase in both productivity and software reliability as the number of terminals made available increased, the study found just the opposite. Both productivity and reliability of software decreased as the ratio of terminals available increased. There was no significance in the results for maintainability (effort to change/effort for repair).

Numerous suggestions have been put forth in attempting to explain this phenomena. Some felt that the increased terminal usage possibly was not properly accompanied with interactive support tools in the particular environment.

Another idea was that the increased terminal availability without proper training for the programmers led to a less disciplined approach by the programmers.

There are several other possible explanations of the results and for that reason, this particular study has been continuing and will be attempting to more thoroughly analyze this data as well as the additional projects that have been completed in this environment.

3.0 Software Testing

A second general set of studies that has been conducted over the past several years within the SEL has been directed toward gaining insight into approaches to testing software. Since this phase of the development life cycle had previously been determined to consume at least 30 percent of the development resources (Reference 5), it was deemed as a critically important discipline to study. Two major experiments were conducted during 1984 and 1985 in an attempt to:

1. Determine the overall coverage of software in the typical testing scenario utilized in the flight dynamics software development.

2. Investigate the relative merits of three standard testing approaches:

- o functional testing
- o structural testing
- o code reading

3.1 Test Coverage* (Chart 6 and Reference 6)

The first experiment on testing was designed to determine the extent to which typical testing techniques within the flight dynamics environment amply exercised the software that had been built. This particular environment utilizes functional testing during both the system test phase as well as the acceptance test phase.

By instrumenting a major flight dynamics system, then by executing the series of both system tests and acceptance tests - experimenters could first determine the coverage attained in the test phases. Next, the experimenters monitored the operational execution of this same software over a period of months to determine the extent to which portions of the completed software were utilized. Finally, the experimenters analyzed uncovered errors in an attempt to determine if the errors occurred in portions of the system that had not been exercised during the

*The lead investigator for this work was Jim Ramsey of Univ. of MD

test phase of development. The software studied was a major subsystem of a mission planning tool and consisted of 68 modules (Fortran subroutines) with 10,000 lines of code. There were 10 functional tests making up the acceptance test plan for the subsystem and during the operational phase, the experimenters monitored 60 operational execution of the software.

3.1.1 Test Coverage Results (Chart 7)

The managers of the flight dynamics development systems noted that the approach to testing had historically been quite good (relatively few errors found in operations) and they expected that the coverage found for this one experiment would be quite high (few modules would be not executed). The results of the experiment showed that for the 10 functional tests executed, only 75 percent of the 68 modules were executed and less than 60 percent of the total executable code was covered in the tests.

Additionally, the series of operational executions showed that a slightly higher percentage of both number of modules and lines of code were executed for this series of 60 executions.

Finally, all of the error reports were reviewed to determine in which portion of the system the errors had occurred. It was found that 8 errors had been recorded during the extended operational phase of the software, but it was found that none of the reported errors occurred in software that had not been executed during the acceptance test phase.

This initial study seemed to indicate that the functional testing approach was properly leading to correct portions of the system being executed and it also was very representative of the operational usage of the software.

The results of this study indicated that further investigations into the various approaches to testing may be worthwhile to determine just which approaches were most effective in uncovering errors in the software itself.

3.2 Software Testing Techniques* (Chart 8 and Reference 7)

Another study was conducted where three programs were seeded with a number of faults and 32 professional programmers from NASA/GSFC and from Computer Sciences Corporation (CSC) participated in an experiment to determine which techniques were effective in uncovering these faults.

The three testing approaches included:

*The lead investigator for this study was Rick Selby of Univ. of MD

- o Functional Testing
- o Structural Testing
- o Code Reading

All programmers participated in applying each of the three techniques.

When performing functional tests, the programmers were required to use the functional requirements along with test results to isolate faults - they were not to look at the source code itself until after testing was completed.

Those programmers performing structural testing used the source code and test results but did not use the functional requirements.

Code reading was carried out with no executions of the software. Those performing code reading reviewed the requirements and also looked at the source code.

3.2.1 Testing Technique Results (Charts 9 and 10)

The results of this experiment indicated that code reading is the most effective of the three testing techniques studied. This technique uncovered an average of 61 percent of all seeded faults while functional testing uncovered 51 percent and structural testing uncovered 38 percent.

Before the test, most of the managers in the SEL felt that code reading would prove to be a very effective testing technique, although they also felt that it would probably be the most costly in manhours to apply; but the results of the experiment indicated that code reading also was the most cost effective technique (3.3 faults per manhour vs 1.8 faults per manhour for structural and for functional testing). It was also noteworthy that, before the experiment, less than 1 out of 4 persons participating in the experiment predicted that code reading would be the most effective approach.

An additional observation that was made after the testing results were compiled was that there seemed to be a difference in the relative effectiveness of each of the testing approaches as the size of the software being tested increased. For the smaller program, code reading was by far the most effective technique, but for the larger program, functional testing seemed to be quite effective. This observation may indicate that there should be a size limit on how much code is utilized in a code reading exercise. Further tests are planned for these studies.

4.0 Software Measures

Over the past 6 to 8 years, the SEL has defined, studied, and evaluated numerous measures applicable to software development and management (References 8, 9, 10). Most of these measures have focused on one phase of the software life cycle - the code/unit test phase. In an attempt to define and apply measures in earlier phases of the life cycle, the SEL has been reviewing several approaches to qualifying or measuring aspects of the software during the specifications phase and during the design phase. Work on the specification phase was reported at the Ninth Software Engineering Workshopand may be foundinreference 11 and 12. One additional piece of work that has been conducted for the design phase will be discussed here.

4.1 Software Design Measures* (Charts 11 and 12 Reference 13, 14)

In an attempt to qualify software designs, a study was conducted to determine if module strength may be utilized as a guideline for software modularizatien. Although the definitions of strength may be well understood, the parameter may not be easy to determine based solely on a structure chart or data flow diagram which may be produced during the design phase of software development.

For the purposes of this study, strength is defined as the 'singleness of purpose' that a software module inherently contains. Singleness of purpose is a subjective parameter assigned at design time by the developer/manager. From a list of potential functionality that a component may have (e.g. computational, control, data processing, etc.) the programmer determines which functions that module contains. High strength would be attributed to those components which have but a single function to perform, medium to 2 and low strength would have three of more functions to perform.

The study examined 450 Fortran modules (from 4 systems) which were built by approximately 20 different developers.

Typical SEL data, which includes detailed cost and error data for all modules was also available for all of the modules. The 450 modules used for this study had a fairly even distribution in size as well as in design strength. Small modules (104 of the 450) were those with up to 31 executable statements, medium (148 of 450) were those with up to 64 executable statements and there were 151 large modules which had more than 64 executable statements.

*The lead investigators for this study were D. Card and G. Page of CSC and F. McGarry of NASA/GSFC

The objective of the study was to determine if strength of modules as determined at design time was related to the cost and reliability of the completed product.

4.2 Results of the Study on Strength (Charts 13, 14, 15)

The results of the study in the SEL indicated that module strength is indeed a reasonable criteria for defining software modularization. When examining the reliability of the 450 modules, it was found that 50 percent of the high strength modules had zero defects while for medium strength modules 36 percent had zero defects and low strength modules only 18 percent of the modules had zero defects. Similar trends were found for the modules of medium error proneness (up to 3 errors per 1000 lines of code) and for modules having a high error rate (over 3 errors per 1000 lines of code).

The distribution of the 'buggy' modules (over 3 errors per 1000 lines of code) was shown to tend more toward low strength as opposed to high strength. Forty-four percent of the buggy modules had low strength while only 20 percent of the buggy modules were found to have high strength.

Several additional observations were made while conducting this particular study. When the characteristics of the individual programmers were reviewed, it was found that those programmers who produced high quality software (low error rate and high productivity) tended to design modules of high strength but they also did not show a preference for writing modules of any specific size. Good programmers generated modules of size that seemed to best suit their design and they did not artificially constrain themselves to writing small modules.

5.0 General Trends and Observations

Over the past several years, the SEL has conducted numerous studies and experiments in an attempt to better understand the impact that various software techniques may have on producing improved software. In addition to the specific studies conducted such as the ones briefly discussed in sections 2, 3, and 4, the SEL has observed general trends in the development and measurement of software. The observations include such points as trends in software reuse, trends in utilization of improved software development technology, and the overall impact of improved developed techniques in the cost and reliability of software over a long period of observation time. Some of these general observations are summarized here.

5.1 Trends in Computer Use and Technology Application (Charts 16, 17)

From data that has been collected on nearly 60 projects over the past 9 years, one trend that has been noted is the tendency to make heavier and heavier usage of available computer support. In 1977 and 1978, computer use averaged approximately 100 runs per 1000 lines of developed source code while in 1982 and 1983 the average use increased to nearly 250 runs per 1000 lines of source. This trend continues to increase within the flight dynamics environment being studied.

Simultaneously, it was noted that the use of more and more structured development practices, improved management approaches and overall higher quality software engineering has continually increased. Each project has been rated on its application of over 200 software techniques (see reference 15) in an attempt to quantify the overall level of development and management technology utilized for a project. The aggregate of the total set of techniques applied results in a rating termed the Software Technology Index. From an average index of less than 100 in 1976 to 1978, it was found that the overall development techniques have increased to an average of over 140 in the 1980's. This seems to point to improved training, better discipline, improved access to tools and possibly better informed management practices.

Although both parameters (computer use and software technology index) seemed to generally increase over the past 7 or 8 years, there is no observed correlation between these two factors.

5.2 Trends in Software Reuse (Chart 18)

Another general observation that was made from the detailed development data collected by the SEL, was that the reuse of software has shown general trends of increase. Typical software systems in the years 1977 to 1979 averaged about 15 or 20 percent reused code while in the 1982 to 1984 timeframe the average reuse has increased to 30 to 35 percent.

Although this reuse is certainly tending in the right direction, the SEL has not conducted detailed studies to determine what the driving factors are in improving the percentage of reuse. The trends are probably indicative of improvements in design technique as well as numerous other factors, but studies have just recently been initiated in the SEL to determine how the trend can be improved at a even faster pace.

It has also been observed in the SEL data that there does not

seem to be a direct relationship between projects that are rated as having a high software technology index and having a high rate of software reuse. But this may not be a surprise since one would expect that high technology usage would lead to follow on systems being able to pick up or reuse software produced by the projects using disciplined approaches for development and management.

5.3 Impact of Development Technologies (Chart 19)

Probably the most basic goal that the SEL has, is to determine the impact that specified software development/management techniques have on the cost and reliability of software. With nearly 60 projects having been closely monitored over the past 8 or 9 years,the SEL attempted to look at general trends inthe reliability and cost of these projects as measured against the software technology index computed for each of these projects. The 200 parameters factored into this index represent everything from structured techniques to disciplined management approaches to configuration control procedures. It is one attempt to characterize each of the projects with a single value.

This technology index correlates very well(r = .82)with reliability of software in the SEL. Those projects with a higher rating of good development practices were the projects with the lower fault rates of the product.

Unfortunately, the impact of this technology index on productivity is quite unclear. The first general observation that has been made is that there is not a clear favorable impact on development cost (cost per line of code) with projects with higher values of this technology index. Studies are continuing in an attempt to more objectively compute this technology rating so that a more conclusive statement can be made. Some researchers also have suggested that it is not to be unexpected that the specific development cost may not decrease but since the reliability has improved and the overall software structure has improved, the maintenance activity will be the beneficiary of the overall cost savings, not the development cost.

5.4 Can Software Technology be Measured? (Chart 20 and Reference 3)

Another major question that software engineers address is whether or not software technology can be measured at all. By utilizing reliability as one major aspect of software quality, the SEL attempted to determine to what extent software development/ management practices could be measured.

There are three levels of development practices which the SEL has hoped and attempted to measure. First, there are individual specific techniques such as the use of structured code or chief programmer team or the use of PDL in design, etc.

Second, there is the usage of a software methodology which is a combination of several methods into a single disciplined approach. This could be the set of methods known as structured techniques which reflect the use of 6 or 8 individual practices such as top down development, structured code, code reading and usage of Unit Development Folders (UDF).

Finally, the attempt has been made to measure the impact of the total technology index which encompasses all disciplined management/development practices. This signifies the level to which the project has attempted to apply recommended software development techniques.

The results of this study indicated:

1. An individual technique cannot be effectively measured in a production environment such as the one in which the SEL is conducting studies. ($r = .37$ is a typical value found in correlating PDL usage and reliability).

2. Disciplined methodologies (combining techniques into a single disciplined approach) can be measured ($r = .65$ for one particular study) and the approaches called Modern Programming Practices (6 techniques) has a significant, measurable, favorable impact on software reliability.

3. Total Software Technology can be measured ($r = .82$ for this one study) and higher levels of applied technology have a marked favorable impact on the reliability of software.

The trends and observations noted here are based on approximately 8 years of data collection and experimentation within the SEL. Approximately 55 projects have been studied and the research is continuing and will continue in the future.

Many of the results are inclusive, but with each experience and study, greater insight is provided into the overall characteristics of the software development process.

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, 86-002, Measuring and Evaluating Software Technology, D. N. Card, F. E. McGarry, J. Valett, to be published

4. McGarry,F.; Valett, J.; and Hall, D., 'Measuring the Impact of Computer Resource Quality on the Software Development Process and Product', Proceedings of the Hawaiian International Conference on Systems Sciences, January 1985

5. McGarry, F., 'What Have We Learned in 6 Years', Proceedings of the Seventh Annual Software Engineering Workshop, December 1982

6. Ramsey, J., and V. R. Basili, 'Analyzing the Test Process Using Structural Coverage', Proceedings of the Eighth International Conference on Software Engineering, August 1985

7. SEL 85-0001, Comparison of Software Verification Techniques, D. Card,R. Selby, F. McGarry, et. al, April 1985

8. SEL, 82-004, Collected Software Engineering Papers: Volume 1, July 1982

9. SEL 83-003, Collected Software Engineering Papers: Volume 11, November 1983

10. SEL 85-003, Collected Software Engineering Papers: Volume 111, November 1985

11. SEL 84-003, Investigation of Specification Measures for the Software Engineering Laboratory, W. Agresti, V. Church, F. McGarry, December 1984

12.Agresti, W.; 'An Approach to Developing Specification Measures; Proceedings from the Ninth Annual Software Engineering Workshop, November 1984

13. Card, D.; Page, G; McGarry, F.; 'Criteria for Software Modularization, Proceedings of the Eighth International Conference on Software Engineering, August 1985

14. Agresti, W.; Card, D.; Church, V.; 'Status Report on Specification and Design Metrics Studies', CSC, December 1985

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

THE VIEWGRAPH MATERIALS

for the

F. McGARRY PRESENTATION FOLLOW

STUDIES AND EXPERIMENTS IN THE SOFTWARE ENGINEERING LABORATORY

December 4, 1985

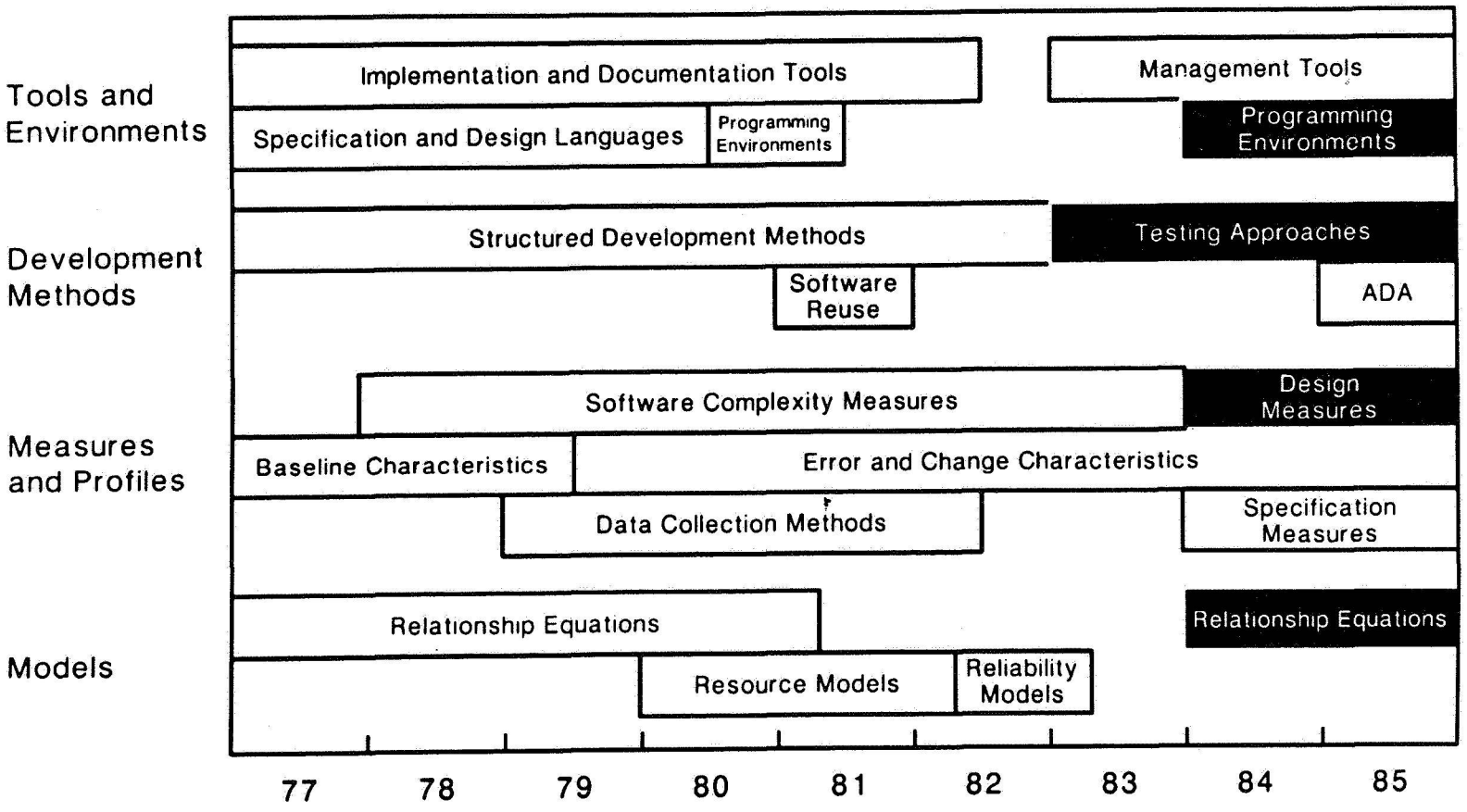
Tenth Annual Software Engineering Workshop

86A0553.04

F. McGarry
NASA/GSFC

CHART 0

SEL RESEARCH TIMELINE



86A0553.13

F. McGarry
NASA/GSFC

CHART 1

MEASURING THE EFFECTS OF ENVIRONMENT ON SOFTWARE DEVELOPMENT

1. Effect of Software Tools (eg. Design Aids, Editors, Auditors)
2. Effect of Overall Computer Support (eg. Turnaround Time, Interactive vs. Batch)
3. Effect of Number of Terminals/ Programmer

ON . . .

- Productivity (LOC/ Man-Month)
- Reliability (Errors/ KLOC)
- Effort to Change (Time to Change Software)
- Effort to Repair (Time to Repair Errors)

86A0553.01

F. McGarry
NASA/GSFC

EXPERIMENT

- 14 Projects of Same General Type in Varying Environments
- Project Size Varies From 11 KLOC to 136 KLOC
- Projects Rated on Various Parameters, Giving Indication of Quality of Environment
- Examined for Correlations Between Ratings and 4 Measures

86A0553.09

ENVIRONMENT VARIATIONS

		Low Rating	High Rating
	Terminals/ Programmer	1/8	1/2
	Number of Tools	2	8
Tools	Tool Usage	Very Low	Very High
	Tool Quality	Very Low	Very High
	Turnaround Time	>1 Day	<2 Hours
Computer	Response Time	>20 Seconds	<5 Seconds
Environment	Batch vs. Interactive	All Batch	All Interactive
	•		
	•		
	•		

86A0553.10

F. McGarry
NASA/GSFC

RESULTS

	Productivity	Reliability	Effort to Change	Effort to Repair
Tool Support	+	0	+	+
Computer Environment	0	0	0	0
Terminals Per Programmer	-	-	0	0

+ = Positive Correlation - for Reliability, Effort To Change, and Effort To Repair - This Implies Lower Errors or Time

- = Negative Correlation

0 = No Correlation

86A0553.07

F. McGarry
NASA/GSFC

TEST COVERAGE

Objective

- Determine Characteristics of Functional Testing in One Environment (Acceptance Testing)
  - % of Code Executed
  - % of Modules Executed
- Compare Acceptance Test Profile With Operational Usage
  - % Code and Modules Executed
  - Profiles of Errors Found

Data for Study

- 1 Flight Dynamics Program
- 68 Modules
- 10K SLOC
- 10 Functional Acceptance Tests
- 60 Operational Use Cases

86A0553.12

F. McGarry
NASA/GSFC

TEST COVERAGE RESULTS

	Acceptance Test	Operational Use
% Code Executed (Total)	56%	65%
% Code Executed (Every Test)	18%	10%
% Module Executed (Total)	75%	80%
% Modules Executed (Every Test)	42%	27%

- 8 Faults Uncovered During Maintenance (Not in Untested Modules)
- Acceptance Tests Were Very Representative of Operational Usage
- For This Environment - Functional Testing Is Good Approach

86A0553.02

F. McGarry
NASA/GSFC

STUDIES OF SOFTWARE TESTING TECHNIQUES

32 Professional Programmers (GSFC and CSC)

3 Expertise Levels: Advanced, Intermediate, Junior

3 Fortran Programs: Seeded With Faults

2 Computers for Online Testing: IBM 4341, VAX 11/780

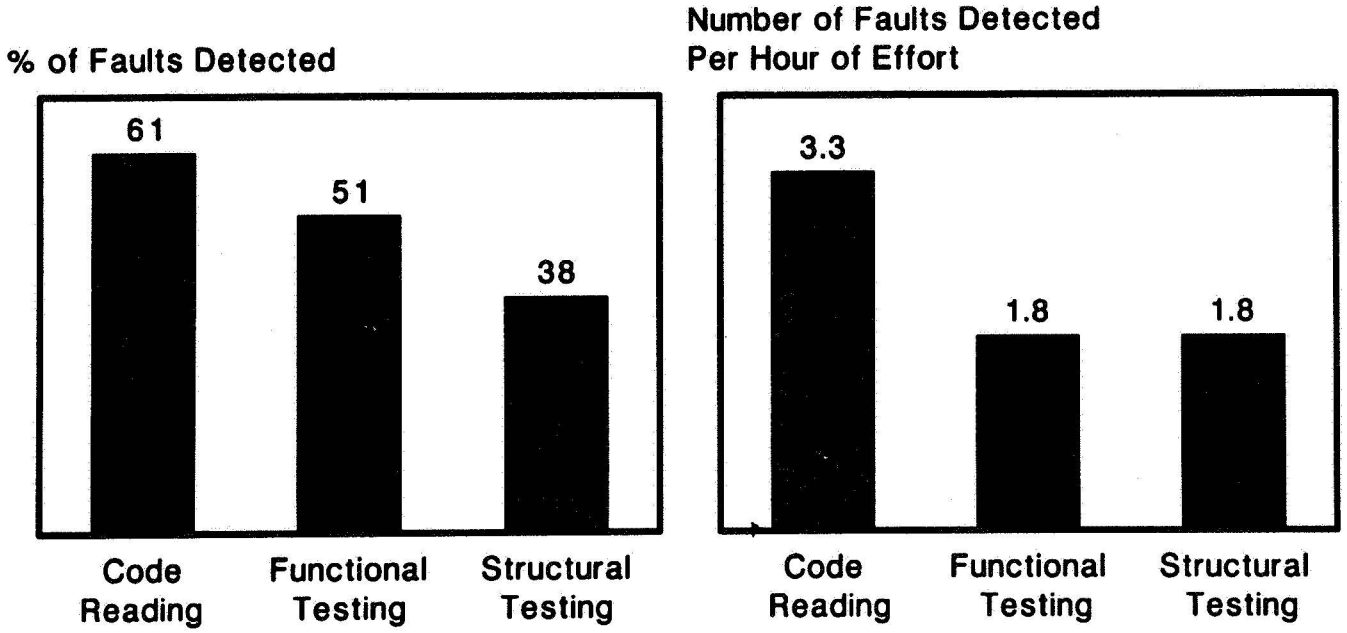
3 Verification Techniques:

	Code Reading	Functional Testing	Structural Testing
View Program Specification	Yes	Yes	After Testing
View Source Code	Yes	After Testing	Yes
Execute Program	No	Yes	Yes

86A0553.11

F. McGarry
NASA/GSFC

SOFTWARE TESTING RESULTS



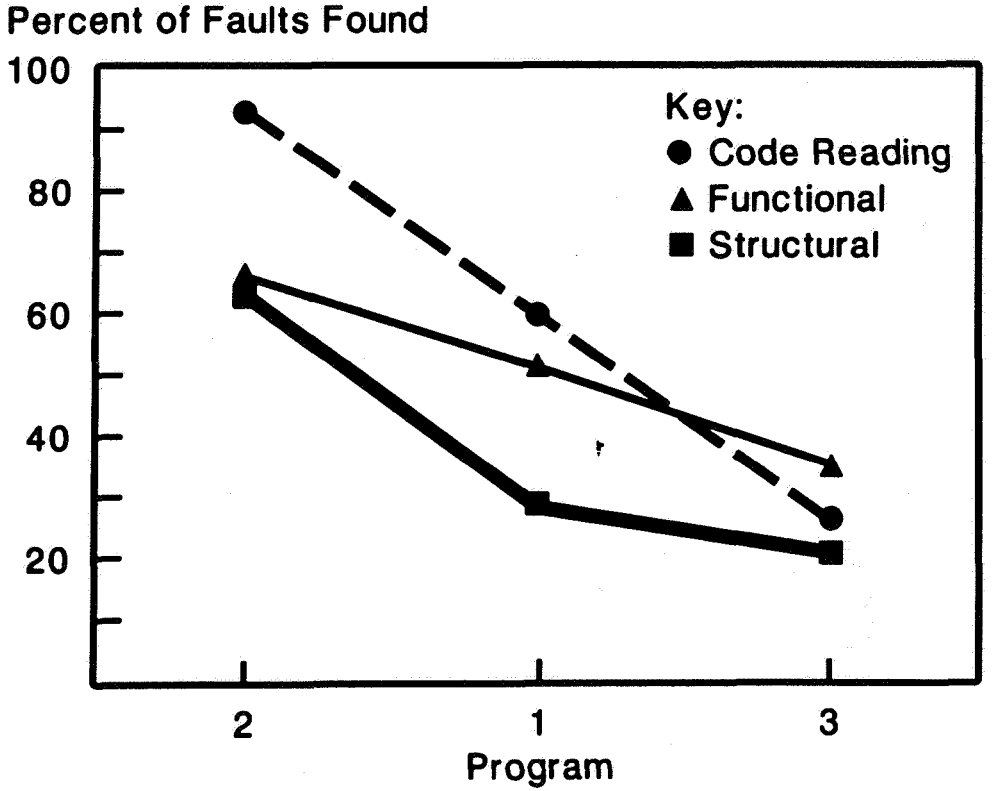
Code Reading Proved To Be the Best Technique in Terms of the Total Number of Faults Detected and the Faults Detected Per Hour of Effort

Prior To the Experiment Only 23% of the Subjects Believed Code Reading To Be the Most Effective Technique

Advanced Subjects Performed Code Reading and Structural Testing Better Than Intermediate or Junior Subjects

86A0553.16

TESTING TECHNIQUES VS. PROGRAM SIZE



Functional Testing May be More Effective for Larger Programs

86A0553.14

Note: Programs Ordered According to Size

F. McGarry
NASA/GSFC

SOFTWARE DESIGN MEASURES

Objective

- Evaluate Strength* and Size as Criteria for Software Modularization

Data for Study

- 450 Fortran Modules
- Approximately 20 Different Developers
- Detailed Cost and Error Data on All Modules
- Detailed Design Descriptions By Programmers at Design Time

* Types (and Numbers) of Functions Performed by Module - Determined by Programmer

86A0553.05

F. McGarry
NASA/GSFC

SOFTWARE DESIGN MEASURES

Strength Distribution

Module Strength	Number of Fortran Modules	Mean Executable Statements	Mean Decisions Per Executable Statement
Low	90	77	0.29
Medium	176	60	0.32
High	187	48	0.32

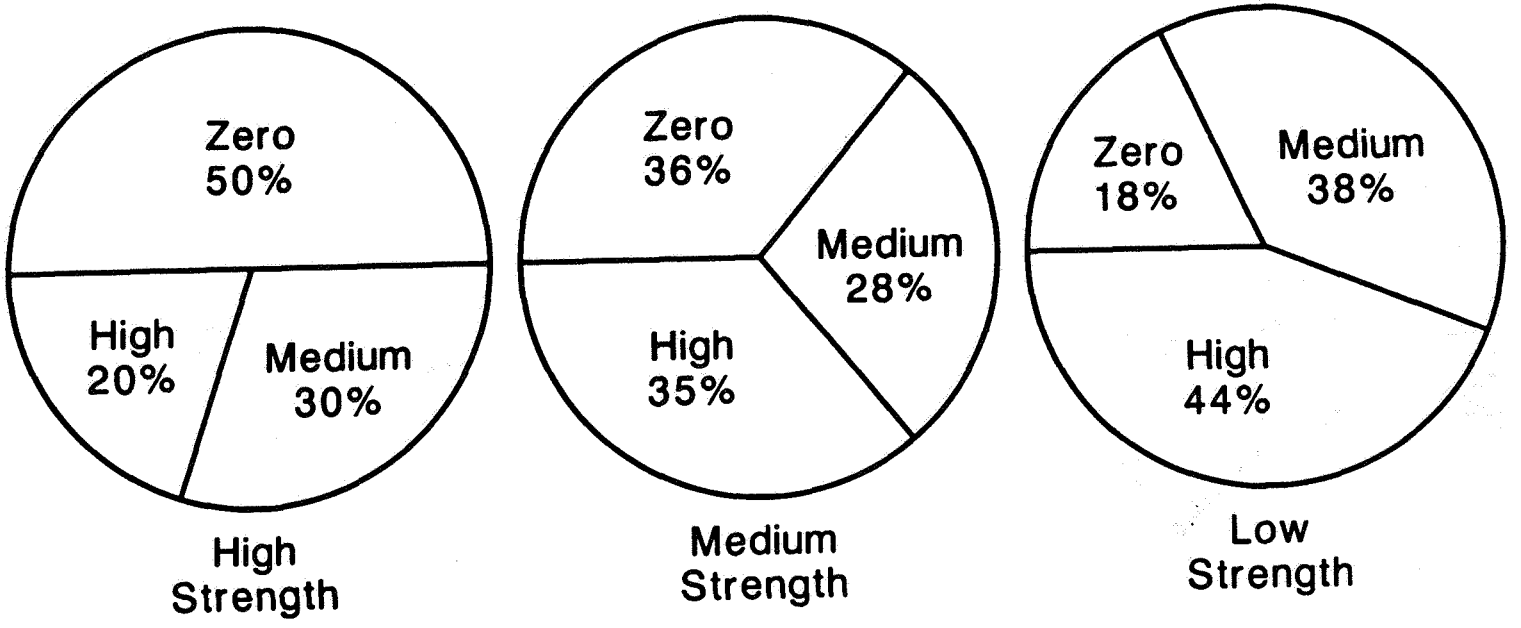
Size Distribution

Module Size	Number of Fortran Modules	Executable Statements	Mean Decisions Per Executable Statement
Small	154	1 to 31	0.31
Medium	148	32 to 64	0.31
Large	151	65 or More	0.32

86A0553.15

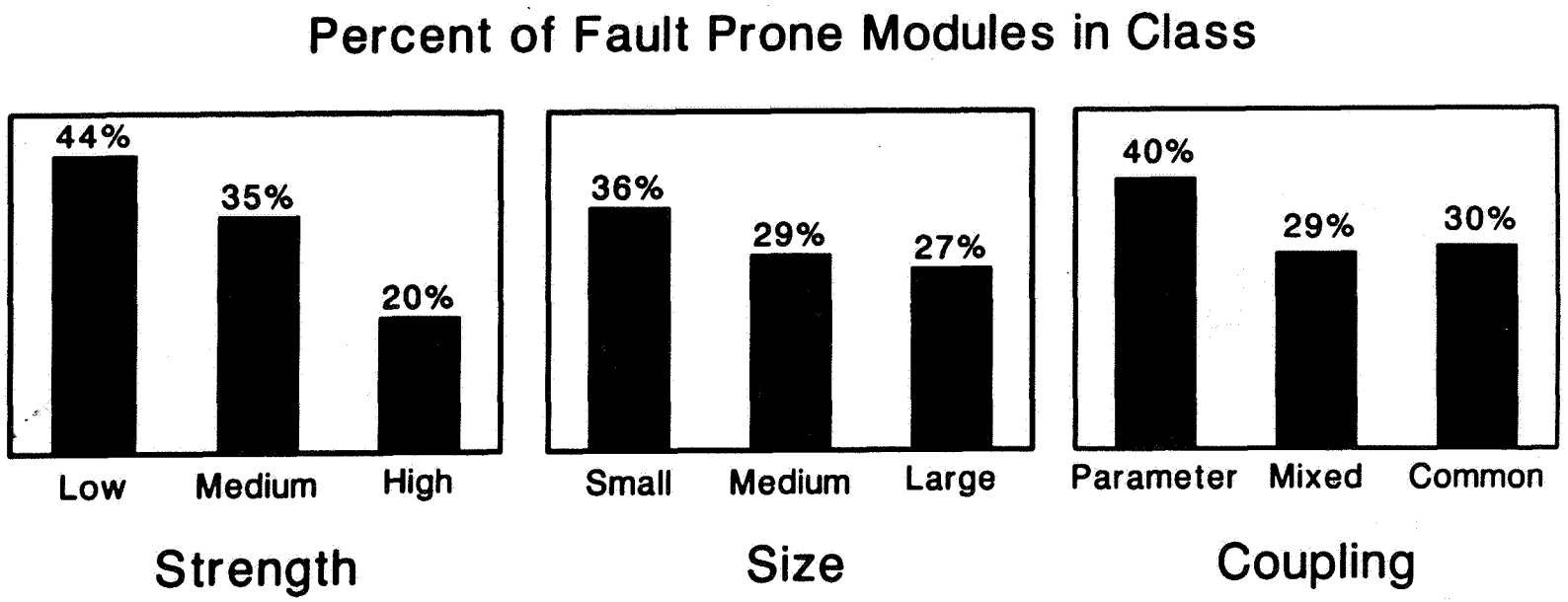
F. McGarry
NASA/GSFC

FAULT RATE FOR CLASSES OF MODULE STRENGTH



86A0553.19

DESIGN CHARACTERISTICS



C-2

86A0553.20

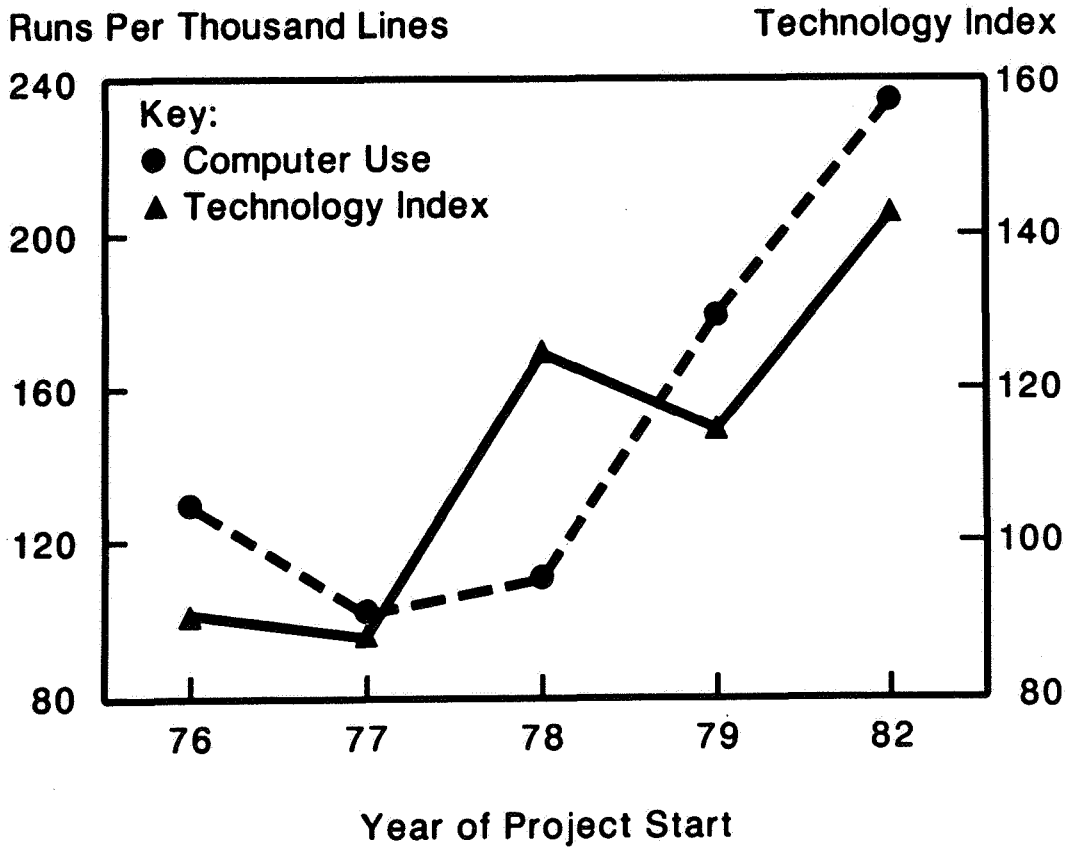
CHART 14

DESIGN MEASURES SUMMARY

- Good Programmers Tend To Write High-Strength Modules
- Good Programmers Show No Preference for Any Specific Module Size
- Overall, High-Strength Modules Have *A Lower Fault Rate and Cost Less* Than Low-Strength Modules
- Overall, Large Modules Cost Less (Per Executable Statement) Than Small Modules
- Fault Rate Is Not Directly Related to Module Size

86A0553.08

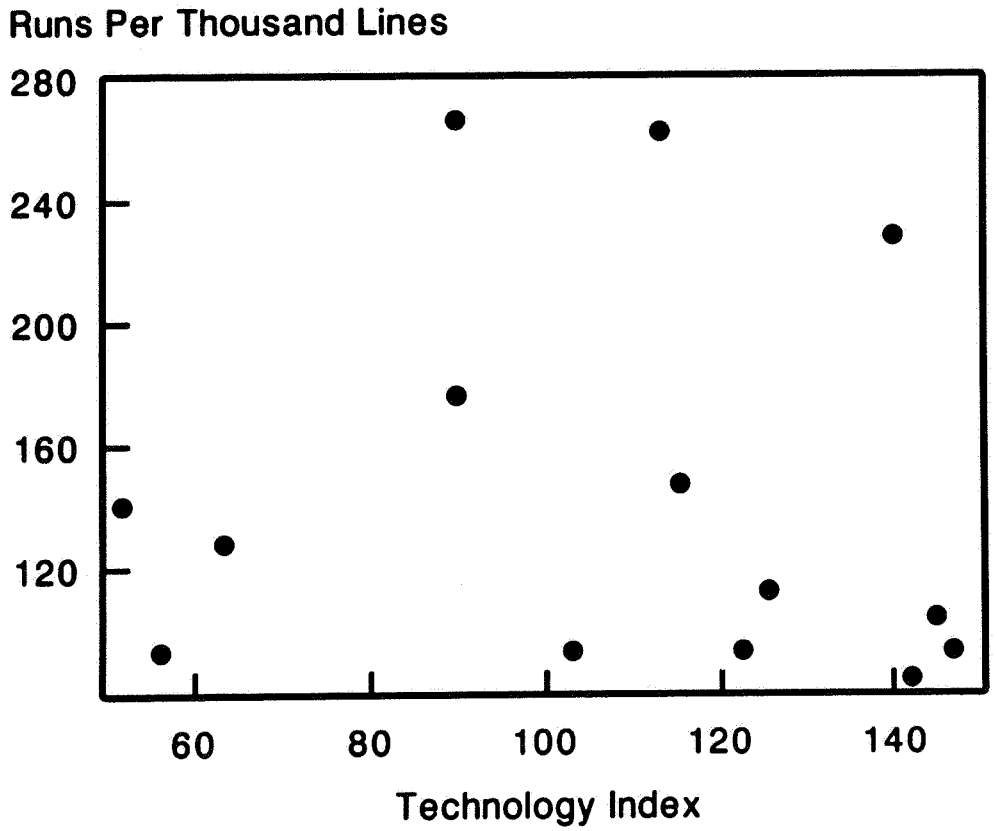
COMPUTER USE AND TECHNOLOGY TIME TRENDS



86A0553.18

F. McGarry
NASA/GSFC

EFFECT OF TECHNOLOGY ON COMPUTER USE

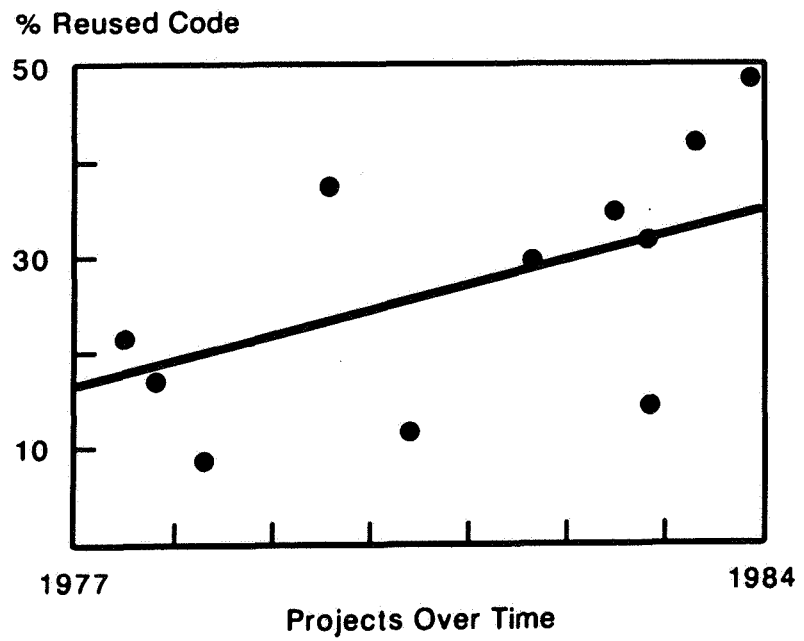


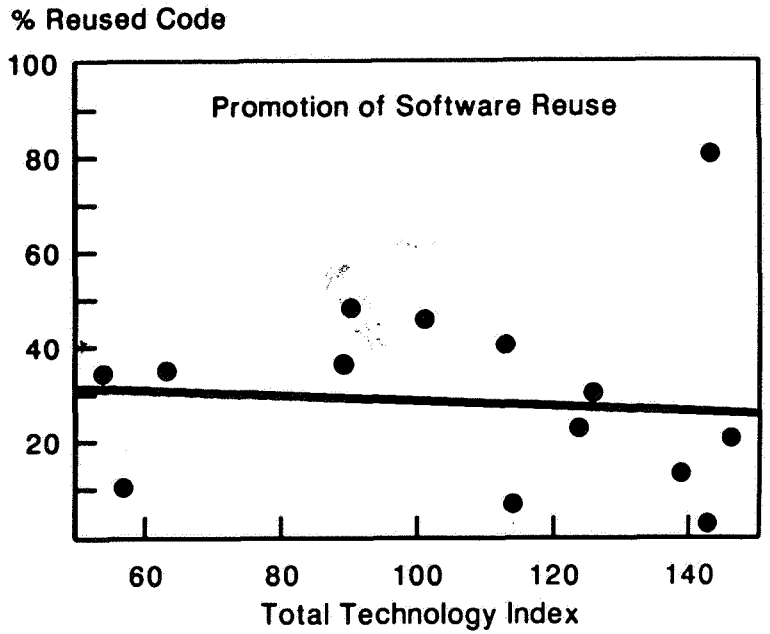
No Obvious Correlation Between Computer Use and Technology Use

86A0553.21

F. McGarry
NASA/GSFC

TRENDS IN SOFTWARE REUSE (BASED ON 15 PROJECTS OF SIMILAR CHARACTERISTICS)



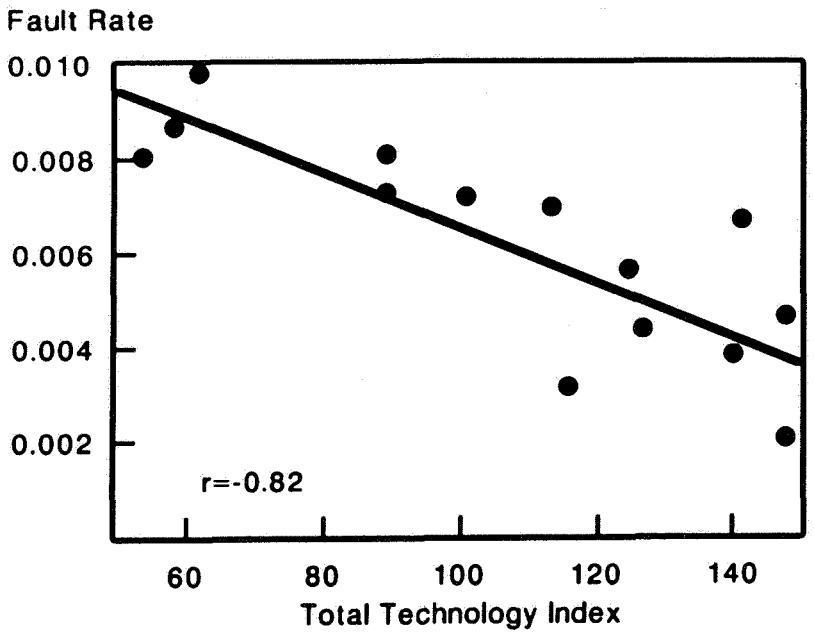


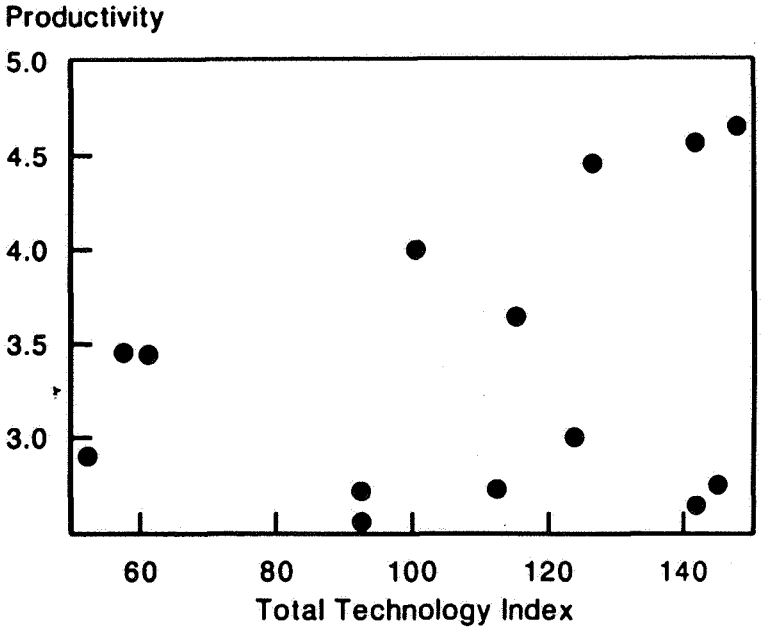
- Software Reuse Has Significant Potential as a "Technology"
- Software Reuse Has Been Increasing Without Directed Efforts
- Currently We Don't Understand Why Software Reused (or Not)

86A0553.22

F. McGarry
NASA/GSFC

EFFECTS OF DEVELOPMENT TECHNOLOGIES

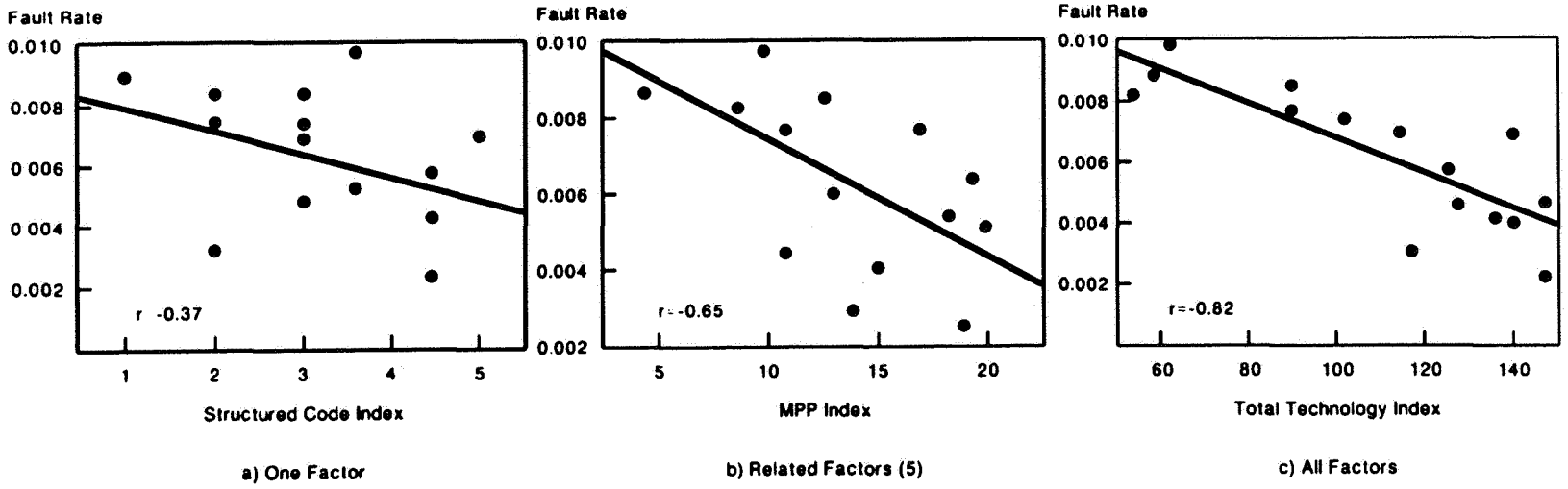




- Reliability Can Be Favorably Impacted
- Productivity Is Sensitive to too Many Other Factors

86A0553.17

EFFECT OF TECHNOLOGY USE ON SOFTWARE RELIABILITY



- Individual Techniques Are Difficult To Measure
- Integrated Methodologies Favorably Impact Quality

86A0553.23