D3-61
30009

MEASURING THE IMPACT OF COMPUTER RESOURCE QUALITY ON THE SOFTWARE DEVELOPMENT PROCESS AND PRODUCT

Frank McGarry and Jon Valett
NASA Goddard Space Flight Center
Greenbelt, Maryland
and
Dr. Dana Hall
NASA Headquarters
Washington, D.C.

ABSTRACT

The availability and quality of computer resources during the software development process has been speculated to have measurable, significant impact on the efficiency of the development process and the quality of the resulting product. Environmental components such as the types of tools, machine responsiveness, and quantity of direct access storage may play a major role in the effort to produce the product and in its subsequent quality as measured by factors such as reliability and ease of maintenance.

During the past six years, the NASA Goddard Space Flight Center has conducted experiments with software projects in an attempt to better understand the impact of software development methodologies, environments, and general technologies on the software process and product. Data has been extracted and examined from nearly 50 software development projects. The data collection and analysis has been performed jointly by NASA, the Computer Science Department at the University of Maryland, and the Computer Sciences Corporation. The projects have varied in size from 3000 up to 130,000 lines of code with an average of 60,000 lines of code. All have been related to the support of satellite flight dynamics ground-based computations. As a result of changing situations and technology, the computer support environment has varied widely. Some projects enjoyed fast response time, excess memory, and state-of-the-art tools, while other projects have endured slow computer response time, archaic tool support, and limited terminal access to the development machine.

This study examined the relationship between computer resources and the software development process and product as exemplified by the subject NASA data. Based upon the results, a number of computer resource-related implications are provided.

BACKGROUND

Computer resource managers have a difficult assignment. They are responsible for selecting and allocating finite computer-related resources in such a manner as to promote an efficient software development process and a resulting quality product. The difficulty is that this job must be accomplished amidst a plethora of potential software tools and a lack of suitable measures for determining the impact of such tools and of other computer resource support.

Numerous measures have been proposed to both characterize the software development process and to quantify and qualify the software product. McCabe (1) defined a measure which could be used to determine the extent of testing that should be performed on a particular software component. This measure, called the cyclomatic measure, has also been used as a means of characterizing the overall complexity of the software. Gilb (2) did a thorough job of describing the practical applications and usefulness of applying software metrics to the software

[46] J. P. J. Kelly, Specification of Fault-Tolerant Multi-Version Software: Experimental Studies of a Design Diversity Approach, UCLA Ph.D. Thesis, 1982.

[47] J. Knight, A Large Scale Experiment in N-Version Programming, *Proc of the Ninth Annual Software Engineering Workshop*, NASA/GSFC, Greenbelt, MD, Nov. 1984.

[48] R. C. Linger, H. D. Mills, and B. I. Witt, *Structured Programming: Theory and Practice*, Addison-Wesley, Reading, MA, 1979.

[49] R. J. Miara, J. A. Musselman, J. A. Navarro, and B. Shneiderman, Program Indentation and Comprehensibility, *Communications of the ACM* **26**, 11, pp. 861-867, Nov. 1983.

[50] H. D. Mills, Mathematical Foundations for Structural Programming, IBM Report FSL 72-6021, 1972.

[51] T. Moher and G. M. Schneider, Methodology and Experimental Research in Software Engineering, *International Journal of Man-Machine Studies* **16**, 1, pp. 65-87, 1982.

[52] G. J. Myers, A Controlled Experiment in Program Testing and Code Walkthroughs/Inspections, *Communications of the ACM*, pp. 760-768, Sept. 1978.

[53] G. J. Myers, *The Art of Software Testing*, John Wiley & Sons, New York, 1979.

[54] T. J. Ostrand and E. J. Weyuker, Collecting and Categorizing Software Error Data in an Industrial Environment, Dept. Com. Sci., Courant Inst. Math. Sci., New York Univ., NY, Tech. Rep. 47, August 1982 (Revised May 1983).

[55] D. J. Panzl, Experience with Automatic Program Testing, *Proc. NBS Trends and Applications*, Nat. Bureau Stds., Gaithersburg, MD, pp. 25-28, May, 28 1981.

[56] D. L. Parnas, Some Conclusions from an Experiment in Software Engineering Techniques, *AFIPS Proc. 1972 Fall Joint Computer Conf.* **41**, pp. 325-329, 1972.

[57] J. Ramsey, Structural Coverage of Functional Testing, *Seventh Minnowbrook Workshop on Software Performance Evaluation*, Blue Mountain Lake, NY, July 24-27, 1984.

[58] Annotated Bibliography of Software Engineering Laboratory (SEL) Literature, Software Eng. Lab., NASA/Goddard Space Flight Center, Greenbelt, MD Rep. SEL-82-006, Nov. 1982.

[59] R. W. Selby, Jr., Evaluations of Software Technologies: Testing, CLEANROOM, and Metrics, Dept. Com. Sci., Univ. Maryland, College Park, Ph. D. Dissertation, 1985.

[60] R. W. Selby, Jr., V. R. Basili, and F. T. Baker, CLEANROOM Software Development: An Empirical Evaluation, Dept. Com. Sci., Univ. Maryland, College Park, Tech. Rep. TR-1415, February 1985. (submitted to the *IEEE Trans. Software Engr.*)

[61] B. A. Shell, The Psychological Study of Programming, *Computing Surveys* **13**, pp. 101-120, March 1981.

[62] B. Shneiderman, R. E. Mayer, D. McKay, and P. Heller, Experimental Investigations of the Utility of Detailed Flowcharts in Programming, *Communications of the ACM* **20**, 6, pp. 373-381, 1977.

[63] E. Soloway, You Can Observe a Lot by Just Watching How Designers Design, *Proc Eight Ann. Software Engr Workshop*, NASA/GSFC, Greenbelt, MD, Nov. 1983.

[64] E. Soloway and K. Ehrlich, Empirical Studies of Programming Knowledge, *Trans. Software Engr.* **SE-10**, 5, pp. 595-609, Sept. 1984.

[65] L. G. Stuckl, New Directions in Automated Tools for Improving Software Quality, in *Current Trends in Programming Methodology*, ed. R. T. Yeh, Prentice Hall, Englewood Cliffs, NJ, 1977.

[66] I. Vessey and R. Weber, Some Factors Affecting Program Repair Maintenance: An Empirical Study, *Communications of the ACM* **26**, 2, pp. 128-134, Feb. 1983.

[67] J. Vosburgh, B. Curtis, R. Wolverton, B. Albert, H. Malec, S. Hoben, and Y. Liu, Productivity Factors and Programming Environments, *Proc. Seventh Int. Conf. Software Engr.*, Orlando, FL, pp. 143-152, 1984.

[68] C. E. Walston and C. P. Felix, A Method of Programming Measurement and Estimation, *IBM Systems J.* **16**, 1, pp. 54-73, 1977.

[69] D. M. Weiss and V. R. Basili, Evaluating Software Development by Analysis of Changes: Some Data from the Software Engineering Laboratory, *IEEE Trans. Software Engr.* **SE-11**, 2, pp. 157-168, February 1985.

[70] L. Weissman, Psychological Complexity of Computer Programs: An Experimental Methodology, *SIGPLAN Notices* **9**, 6, pp. 25 - 36, June 1974.

[71] S. N. Woodfield, H. E. Dunsmore, and V. Y. Shen, The Effect of Modularization and Comments on Program Comprehension, Dept. Com. Sci., Arizona St. Univ., Tempe, AZ, working paper, 1981.

development process. He described methods of estimating errors, cost, and overall software quality as well as approaches for applying factors such as methodology and environment to cost estimation. Basili and Freburger (3) characterized various relationships between software size, effort, reliability, and quality. Their work helped to verify some earlier efforts which had been done by Walston and Felix (4) and by Jeffrey and Lawrence (5). These studies indicated a near linear relationship between effort and lines of code. The Basili work produced the relationships:

$$E = 1.55 * (NL**.99) \text{ and } E = 1.48 * (DL**.98)$$

where E = effort, NL = new lines of code, and DL = developed lines of code. Later, Bailey and Basili (6) described a mechanism for refining local resource estimation models with locally tuned parameters. They describe the process by which the cost equation, which is based exclusively on lines of code, could be refined by solving for and adding such factors as the computer support environment or the personnel experience on the overall development process. Putnam (7) noted that the commonly used metrics of productivity and cost per instruction have been found to be very sensitive to schedule variation, system size, and the development environment. Putnam went on to develop mathematical relationships that express software development progress in accordance with Shannon's information theories. Specifically developed is a "software equation" that relates executable delivered source statements and three critical parameters: the life cycle effort in manyears, the implementation project duration in years, and a parameter (C_k) that attempts to account for quantized technology and development environment effects. Putnam's software equation is:

$$Ss = C_k K**1/3(t**4/3)$$

It is commonly thought that the availability of good development tools in an integrated reliable support environment will materially contribute to software with quality attributes such as those defined by McCall. An industry example where such thinking is being put to practice is the Software Productivity Project initated in 1981 at TRW (8). The major thrust of that project is the establishment of a software development environment that includes among other aspects, the provision for immediate access to computing resources and integrated tool sets. Although still early in the TRW program, productivity gains to date have been quite encouraging. In one instance, for example, people were moved out of the experimental productivity environment back into the traditional TRW surroundings. Of the 23 possible respondents, 20 indicated that their productivity in the improved environment had been almost 50 percent higher. The dominant features of that gain were the availability of software tools to support software development and office functions (approximately 15 percent), a personal terminal with high speed access to computers (roughly 14 percent), and a private office with modern office furniture (about 8 percent).

At present, it is not known whether gains such as the above are universally possible or are peculiar to certain types of projects or situations. The Software Engineering Laboratory (SEL) at the NASA Goddard Space Flight Center (GSFC) is actively pursuing answers to such questions. Created in 1977, the SEL has three major goals:

1. Improve the understanding of the software development process in the time varying environment;
2. Intentionally instill and measure the effects of various methodologies, tools, and models; and
3. Identify and subsequently apply successful practices.

The SEL is funded and coordinated by NASA/GSFC and is supported by the Computer Science Department at the University of Maryland and by Computer Sciences Corporation. The SEL functions as part of the organization at Goddard which develops the flight dynamics ground support software for Goddard missions. Within this production environment, the SEL experiments by applying varying degrees of proposed techniques (such as development methodologies and tools) then studying the resulting product to determine the resulting impact. Detailed software development data is collected throughout the entire development life cycle as described in references 9 and 10.

To date, the SEL has monitored nearly 50 development projects totaling over 2 million lines of source code.* The projects have ranged in size from 3,000 to 130,000 lines of code with the average project running about 60,000 lines. This relatively unique effort (at least within NASA) is beginning to yield substantial returns; one of which is a better understanding of the importance of the computer support environment on the production of reliable, cost effective software.

*Line of code = 80 byte record processable by compiler or assembler. It includes commentary and other non-executable lines.

By the term "computer support environment", we mean the cumulative availability of software tools and support other than tools, e.g., terminal availability, job turnaround time, and storage capacity. In the light of this definition, the computer support environment within the SEL has varied from very poor to very good.

In order to compare projects from these environments, certain measures have been adopted by the SEL (and this investigation) as representative of process efficiency and product quality. These measures are productivity, reliability, and two measures of maintainability: effort to change and effort to repair. These terms are defined as:

- Productivity - the amount of product in lines of source code that is output per person-month invested,
- Reliability - the number of errors per thousand lines of source code,
- Effort to Change - the average amount of time needed to effect a non-error change, and
- Effort to Repair - the average amount of time required to make an error correction.

Reliability and both maintenance factors are measured from the time that unit (module) coding and testing is considered to be complete.

The relationship explored in this paper is that between the development environment (as measured by tools and support other than tools) and the dual attributes of process efficiency and product quality (as indicated by the above four factors). Specifically, the following questions are addressed:

1. Does the availability and increased usage of tools have a favorable, measurable impact?
2. Do other, non-tools types of computer resource support play a determinable role?
3. Does increasing the ratio of terminals to programmers have a positive, measurable influence?

EXPERIMENT APPROACH

Fourteen flight dynamics software projects were selected for examination in this study. These projects were chosen because they ranged across a number of computer support environments, data had been recorded in the SEL data base, the projects were of similar complexity and utilized similar teams of people, and because the responsible two project managers were available to provide subjective ratings and opinions when needed.

The approach used in this experiment consists of four steps:

Step 1 - Data descriptive of the selected development efficiency and product quality measures were compiled from the SEL data base. This data is presented in Table 1. Indicated as well is the size of each project in lines of code. This data shows considerable variation between projects. Reliability (errors per KLOC), for example, varies from 0.3 up to 10.

Step 2 - The types of non-tool computer resource support shown in Table 2 were selected. The two managers that had individually led the 14 projects then subjectively scored the amount of use of each type of non-tool support on a scale from 1 to 3 for each project. Two non-tool support measures were used in the later statistical correlations. One was the individual factor, ratio of terminals to programmers, and the other was the simple total non-tools score for each project (see the last column of Table 2). The terminals to programmers factor was singled out because of its intuitive probable importance.

Step 3 - This step consisted of first identifying twenty one software tools potentially available to the selected projects and then rating the use and quality of those tools. As presented in Table 3, the specific tools are classified as assemblers and compilers, documentation and configuration control aids, debuggers, design tools, editors, preprocessors, development and post-development aids, and requirements tools. These tools were subjectively ranked by the project managers using a scale of 0 to 3 (see Table 4). The tools ratings then were subjected to further adjustment by factoring in four tool quality measures: reliability, level of integration into the work environment, ease of use, and tool usefulness. These subjective scorings are shown in Table 5. The resulting adjusted tool measures (= usage x quality) were summed to give a tools score for each project. No weighting of the tools was applied, so that although a compiler may be more important to the overall project than is an accounting aid, both carried equal weight in this analysis.

Step 4 - After all the data was compiled, a means for determining the relationships between the defined computer support environment and the selected project quality and process efficiency measures was needed. The Wilcoxon Two Sample Test [11] was determined to be the proper statistical method for studying these relationships. An example of an application of this

technique is shown in Table 6. First, the projects were divided into two groups by choosing a breaking point between high tool support and low tool support. That is, those projects with an aggregate tool support number over a certain value were considered high tool support, while those under that value were considered low support. Then the productivity values for each project were ranked from best to worst. The sum of these ranks was used to determine the significance of the correlation between tool support and productivity. In this case, since the smaller group (low tool support) has a higher total, i.e., productivity was worse in the low tool support projects, a correlation exists between high tool support and productivity. The Wilcoxon Test indicates that there is an 11.4 percent chance that the correlation between high productivity and high tool support is random. Similar tests were performed to study each of the environmental factors affects on each of the quality measures.

TABLE 1
Development Efficiency and Product Quality Measures for
14 Selected Projects

Project	Size (KLOC)	Loc / MDAY	Errors / KLOC	Effort to Repair*			Effort to Change*		
				Low	Med	High	Low	Med	High
A	46	22	0.3	2	3	12	172	80	18
B	46	27	3.9	118	57	6	57	35	14
C	54	27	0.5	14	10	3	4	5	10
D	49	17	9.4	239	176	55	180	184	130
E	49	18	8.9	234	175	37	136	94	86
F	136	21	4.3	401	148	34	218	174	80
G	78	36	3.9	182	97	20	152	154	98
H	67	31	4.5	162	133	10	146	110	22
I	22	8	10	190	32	1	152	57	22
J	11	30	1.6	4	11	3	1	11	2
K	46	38	0.9	15	17	10	8	23	27
L	30	10	1.1	17	16	1	9	21	20
M	15	11	5.5	33	41	9	18	26	11
N	17	10	5.6	27	41	30	38	57	28

*Entries from left to right indicate number of repairs or changes requiring low effort (1-2 hrs.), medium effort (2-8 hrs.), high effort (more than 8 hours).

EXPERIMENT RESULTS

The results of this limited experiment show a great deal of variance over the affects of computer support environment on the efficiency of the software development process and the quality of the resulting software product. Some significant relationships between overall tool usage and the defined measures were found, but the impact of support other than tools did not prove to be obvious. Some of the results were quite surprising.

TOOL USE AND PRODUCTIVITY:

A significant correlation was found between increased tool usage and increased productivity.

It had been hypothesized that greater usage of software tools during the software development process would result in an increase in software productivity. A strong relationship supporting that hypothesis was found. The probability that the positive relationship is due to chance is 11 percent. However, no attempt has been made in this investigation to determine the magnitude of the productivity gain. The TRW study previously referenced indicated that greater usage of software tools was responsible for nearly a 15 percent productivity increase. Further analysis is necessary to quantify the corresponding SEL estimate.

TOOL USE AND RELIABILITY:

No significant correlation was found between increased tool usage and increased software product reliability.

The fourteen projects in this study reported error rates after unit testing completion ranging from 0.3 to over 10 errors per KLOC. Prior to the statistical analysis, it was anticipated that the projects with the lower error rates would be those where

TABLE 2
Application of Non-Tool Support to
Development of Selected Projects

Projects	Interactive Rather Than Batch	Terminal Accessibility	Terminals Per Programmer	Compiler Speed	Turnaround Time	Avg. CPU Utilization	System MTTF	System MTTR	Nearness	Direct Storage	Offline Storage	Project Total
A	1	1	1	3	1	2	1	1	1	1	1	14
B	1	1	1	3	1	2	1	1	1	1	1	14
C	1	1	1	3	1	2	1	1	1	1	1	14
D	2	2	2	3	1	2	1	1	1	1	2	18
E	2	2	2	3	1	2	1	1	1	1	2	18
F	3	3	3	3	2	2	3	3	1	3	3	29
G	1	1	1	3	1	2	1	1	1	1	1	15
H	2	1	1	3	1	2	1	1	1	1	1	16
I	3	3	3	3	3	2	3	3	2	3	2	30
J	3	3	3	3	3	3	3	3	3	3	2	32
K	3	2	2	3	2	3	3	3	2	3	2	28
L	3	3	2	3	2	3	3	3	2	3	2	29
M	3	2	3	1	1	1	3	3	2	1	2	22
N	3	2	3	2	1	1	2	2	2	1	2	21

Notes: a.) 1 = low or poor
2 = medium or average
3 = high or good
b.) 1 for Avg CPU utilization implies CPU highly utilized
c.) 1 for MTTF implies short intervals between failures
d.) 1 for MTTR implies long time waiting for repairs.

TABLE 3
Spectrum of Tool Support

Assembler/Compilers: PDP INTEL VAX 4341 360	Documentation and Configuration Control: Source Code Management Tool Software Documentation (SDOC) Gessdoc Configuration Analysis Tool (CAT)
Debugging: Symbolic Debugger	Design Aids: Program Design Language
Editors: VAX 360 4341	PreProcessors: Namelist Preprocessor (NPP) Structured Fortran Preprocessor (SFORT)
Development and Post Development Aids: General Accounting Aids Timing and Program Optimizer Data Simulator (data generator) Source Analysis Program (SAP)	Requirements Tools: Automatic OnLine Requirements Analyzer

TABLE 4
Tool Usage Rating for Selected Projects

Projects	Assemblers/ Compilers					Source Code Mgmt	SDOC	Gessdoc	CAT	Symbolic Debugger	PDL	NPP	SFORT	Accounting	Timing/ Optimizer	Data Simulator	SAP	Req'ts/ Analyzer	Editors		
	A	B	C	D	E														A	B	C
A	0	0	0	0	3	3	3	2	0	0	1	0	1	2	2	3	2	0	0	1	0
B	0	0	0	0	3	3	3	3	0	0	1	0	2	2	2	2	2	0	0	1	0
C	0	0	0	0	3	3	3	2	0	0	1	0	2	2	2	2	2	0	0	2	0
D	0	0	0	0	3	3	1	3	3	0	1	0	3	2	2	3	2	2	0	2	0
E	0	0	0	0	3	3	1	3	2	0	1	0	3	2	2	3	2	2	0	2	0
F	0	0	0	3	0	3	0	2	3	0	1	0	3	3	0	3	2	1	0	0	3
G	0	0	0	0	3	3	1	3	0	0	1	0	3	2	2	2	2	0	0	2	0
H	0	0	0	0	3	3	1	3	0	0	1	0	3	2	2	2	2	0	0	2	0
I	0	0	3	0	0	0	0	0	1	2	1	2	0	1	0	1	2	1	2	0	0
J	0	0	3	0	0	0	0	0	1	2	1	3	0	1	0	1	2	1	3	0	0
K	0	0	3	0	0	0	0	0	1	2	1	3	0	1	0	1	2	1	3	0	0
L	0	0	3	0	0	0	0	0	1	3	1	1	0	1	0	1	2	1	3	0	0
M	0	3	0	0	0	0	0	0	1	3	1	3	0	1	0	2	2	1	3	0	0
N	3	0	0	0	0	0	0	0	1	0	1	3	0	1	0	2	2	1	3	0	0

0 = Tool Not Available
1 = Tool Available but Used Very Little
2 = Tool Available and Used Somewhat
3 = Tool Available and Used Extensively

TABLE 5
Tool Quality Matrix

	Assembler/ Compilers					Documentation & Config. Ctrl.				Symbolic Debugger	PDL	Pre-Processors		Development Aids				Req'ts/ Tools	Editors		
	A	B	C	D	E	Source Code Mgmt	SDOC	Gessdoc	CAT			Namelist	SFORT	Accounting	Timing/ Optimizer	Data Simulator	SAP		A	B	C
Reliability	5	1	5	5	5(1)	5	5	5	4(2)	5	5	5	5	5	4	4(3)	5	3	5	5	5
Level of Integration	5	1	5	5	5	4	4	4	1	4	4	3	4	5	2	2	1	2	4	4	4
Ease of Use	3	1	5	4	4	5	5	5	4	4	4	5	5	5	1	2	2	1	5	4	4
Functionality (usefulness)	5	5	5	5	5	4	2	3	3	4	4	3	5	2	2	5	2	2	4	4	4
Totals	18	8	20	19	19(1)	18	16	17	12(2)	17	17	16	19	17	9	13(3)	10	8	18	17	17

Notes: 1 - 5 except for Project A which equals 2 and column total = 16
2 - 4 except for Projects D and E which equal 2 and column total = 10
3 - 4 except for Project M which equals 2, Project C = 3, and Project B = 1

TABLE 6
Example Application of Statistical Correlation Technique

High Tools Support Projects	Tools Score	Productivity	Productivity Rank
A	332	22	7
B	374	27.1	6
C	378	27.7	5
G	384	36	2
H	384	31	3
F	424	21	8
E	433	18	9
D	443	17	10
			50 Total

Low Tools Support Projects	Tools Score	Productivity	Productivity Rank
I	249	8.8	14
N	256	10.3	12
L	268	10	13
M	273	10.9	11
J	273	30	4
K	283	38	1
			55 Total

the tool usage was the greatest. However, the data showed no such measurable relationship. (The probability that the exemplified relationship was simply due to random chance was 42 percent.)

TOOL USE AND MAINTAINABILITY:

A significant correlation was found between increased tool usage and software maintainability.

The SEL maintains in its historical data base detailed records of all the modifications made to a project's software after the units comprising that software have completed unit testing. The analyses made on the data found a high correlation between increased tool usage and ease of repair (only a 9 percent chance that the correlation is random) and between increased tool usage and ease of change (4 percent possibility of random chance). Such findings point to a very favorable impact on the quality of the software product when the development is aided with increased application of software tools.

NON-TOOLS SUPPORT AND ALL FACTORS:

No significant correlation was found between the cumulative contributions of other, non-tools forms of computer resource support and any of the four defined measures.

The overall effect of non-tools types of computer resource support on the four measures seems to be minimal, at least according to the data used in this investigation. In fact, a disturbing finding was that an inverse relationship appears to exist between ease of change of the end product and increased non-tools computer resource support. If true, this means that greater numbers of terminals, more ready access to printers, faster compilers, etc. result in software that is more difficult to modify. The probability that this finding is due to random chance was only 14 percent. Needless to say, this finding is receiving considerable attention at present in the SEL.

RATIO OF TERMINALS PER PROGRAMMER AND ALL FACTORS

The data available to this analysis indicated a negative correlation between the number of terminals per programmer and productivity and reliability.

The most unanticipated result was that as the availability of terminals increased, the efficiency of the process and the quality of the product seemed to decrease. The relationship between lower terminal availability (more programmers per terminal) and productivity resulted in a correlation value of slightly less than ten percent. The relationship between lower terminal availability and reliability was even more pronounced, indicating a less than 5 percent chance that the statistic was due to chance.

Obviously, this result caused disbelief and many rechecks of the compiled data. Two plausible explanations are:

1. The sample size is too small,
2. Programmers accustomed to a terminal shortage are disciplined to be better prepared by doing more design and checking at their desks. Programmers with an abundance of terminals may not do the critical desk preparatory work and instead try to design, test, enhance, etc. directly at the terminal.

Investigations are continuing to better understand this result and its implications.

CONCLUSIONS

Based upon the findings of this study, it appears that managers can gain worthwhile benefits by concentrating on providing software development tools for their programmers and not focus as much attention on other, non-tools types of computer resource support, e.g., improved terminal and computer accessibility, faster compilers, and more online storage.

This research indicates the following answers to the three questions posed earlier in this paper:

Q: Does the availability and increased usage of tools have a favorable, measurable impact?

A. The SEL data shows a significant correlation between increased tool use and productivity and between tool use and product maintainability.

Q: Do other, non-tools types of computer resource support play a determinable role?

A: Non-tools types of computer resource support do not seem to have much influence on the software development process or the quality of the resulting product, at least within the SEL.

Q: Does increasing the ratio of terminals to programmers have a positive, measurable influence?

A: Unexpectedly, the data shows that more terminals may have a detrimental effect on the quality of the resulting software and the efficiency of the development process. A significant inverse relationship was found in the data recorded for the 14 subject projects.

It must be emphasized that these resuits may not be applicable to other software development environments. The tool set utilized in this study was limited as was the variation in the overall software development environment. Although the results may not be universally true, the analysis technique should be valid and may serve as a model for similar investigations elsewhere.

Research in the general topic area of software tools and other forms of computer resource support is continuing within the SEL. A future goal is to quantify the productivity and quality gains and losses in order to provide more substantiative guidance to managers and to enable comparisons to other research efforts such as the Software Productivity Project at TRW.

References

1. McCabe, T., "A Complexity Measure", IEEE Transactions on Software Engineering, Vol. SE-2, Number 4, December 1976.
2. Gilb, T., "Software Metrics", Winthrop Computer Systems Series, Gerald Weinberg, editor, 1977.
3. Basili, V. R., K. Freburger, "Programming Measurement and Estimation in the Software Engineering Laboratory", The Journal of Systems and Software 2, pg. 47-57, 1981.
4. Walston, C. E., C. P. Felix, "A Method of Programming Measurement and Estimation", IBM System Journal 16, 1977.
5. Jeffrey, D. R., M. J. Lawrence, "An Internal Organizational Comparison of Programming Productivity", Department of Information Systems, University of New South Wales, 1979.
6. Bailey, J. W., V. R. Basili, "A Meta-Model for Software Development Resource Expenditures", Proceedings, Fifth International Conference on Software Engineering, 1981.
7. Putnam, L. H., "The Real Metrics of Software Development", 1980, EASCON Proceedings, IEEE.
8. Boehm, B. W. et al., "A Software Development Environment for Improving Productivity", Computer, June 1984.
9. McGarry, F. E., G. Page, et al, "The Software Engineering Laboratory", NASA Technical Report, SEL 81-003, September 1981.
10. Church, V., D. Card, F. McGarry, "Guide to Data Collection", NASA Report SEL-81-001, September 1981.
11. Marascuilo, L. A., M. McSweeney, "NonParametric and Distribution-Free Methods for Social Sciences", 1977, Brooks/Cole Publishing Company, Monterey, California, pg. 267-273.

Biographies

DR. DANA L. HALL

Currently Level A Manager for Data Systems and Software for the NASA Space Station program. Previously associated with NASA's office of the Chief Engineer in programs to improve NASA software management practices and to standardize space data system operational elements (telemetry, telecommand, etc.) across all space agencies. Eight years industrial experience with the MITRE Corporation and TRW.

FRANK E. MCGARRY

Mr. McGarry is Head of the Systems Development Branch at NASA/Goddard Space Flight Center where he is responsible for directing the software development of flight dynamics systems. The Branch is also responsible for conducting research in software engineering technology which is to be applied to the operational software systems. He has been with Goddard for 18 years.

JON D. VALETT

Currently working in the Systems Development Branch at Goddard Space Flight Center, where he is responsible for both the development of applications software and software engineering research. Joined Goddard in 1983, immediately after graduating from the University of Iowa.