N87-25776

A SOFTWARE TECHNOLOGY EVALUATION PROGRAM

AUTHOR: DAVID N. NOVAES-CARD

Adress to contact: Computer Sciences Corporation
8728 Colesville Road
Silver Spring, Maryland 20910
USA
Telephone (301) 589-1545

INFORMÁTICA 85

KEYWORDS: Software Engineering, Productivity, Reliability, Maintainability, Modern Programming Practices, Quality Assurance, Computer Use.

INTRODUCTION

Software provides an increasingly larger part of the functionality, and consequently the cost, of computer systems. For some large applications, the cost of the software component exceeds 75 percent of the total system cost. The ability to deliver reliable software on time at minimum cost has become essential to success in the computer industry. The delayed first launch of the National Aeronautics and Space Administration (NASA) space shuttle clearly demonstrated the consequences of software failure.

Software development organizations, therefore, have strong incentives to improve the software development process, principally by adopting new technology. A wealth of potentially beneficial software engineering tools, practices, and techniques has emerged in the past several years. Many, nowever, have been empirically evaluated (Reference 1). Furthermore, experience shows that all software engineering technologies are not appropriate for all software development problems and environments.

The difficulty of accurately measuring the software development process, in general, and technology use, in particular, accounts for much of the lack of objective information in this area. This paper describes an ongoing technology evaluation program (Reference 2) conducted by the Software Engineering Laboratory (SEL) that is intended to resolve these issues, at least in part. In the context of this paper, the term "technology" refers to tools, practices, and technique applied by software developers.

Software engineering laboratory

The SEL is a research project (Reference 3) sponsored by NASA and supported by Computer Sciences Corporation and the University of Maryland. Figure 1 shows the organization of the SEL, which was established in 1977. The SEL studies software developed to support spacecraft flight dynamics applications at Goddard Space Flight Center.

The overall objective of the SEL is to understand the software development process in the flight dynamics environment and the identify the ways in which it can be altered to improve the quality and reduce the cost of the product. The SEL has monitored the development of more than 45 fligth dynamics projects. In addition, the SEL conducts controlled experiments and performs multiproject variation studies.

Flight dynamics software

The general class of spacecraft flight dynamics software studied by the SEL includes applications to support attitude determination/ control, orbit adjustment, maneuver planning, and mission analysis. The attitude ground support systems form a large and homogeneous group of software that has been studied extensively. Each system includes a telemetry processor, data adjuster, and attitude computation subsystems as well as other necessary suporting functions.

Flight dynamics applications are developed in FORTRAN on IBM mainframe computers. System sizes range from 30 to 150 thousand source lines of code. The fixed spacecraft launch date imposes a severe development time constraint. Acceptance testing must be completed 2 months prior to launch so that launch preparations can proceed on schedule. Figure 2 describes some major characteristics of flight dynamics software.

THE PROGRAM

The SEL program of technology evaluation includes three steps: measurement, evaluation, and transference. Measurement establishes the baseline against which the effects of technologies can be compared. Next, technological innovations are attemped and their effects evaluated. After careful study, sucessful technologies are transferred to developers via guidelines, standards, and training.

Measurement

Measurement is the basic prerequisite for technology evaluation and management. Software engineering experts such as Boehm (Reference 4) and DeMarco (Reference 5) are paying increased attention to the role of measurement in software development. The SEL developed a comprehensive data collection methodology (Reference 6) as the basis for its measurement activity. Measures collected include staffing, computer utilization, error reports, and product size/complexity measures, as well as the level of technology applied to each project. The SEL employs both questionnaires and automated methods of data collection. The collected data are assembled in a computerized data base accessible to all SEL participants.

Because the software development process is complex and involves many different human and physical elements,

many measures are needed to characterize it adequately. Examination of a model of software development, such as that shown in Figure 3, helps to define overlapping set of measures. This model includes the following components:
- Problem — Statement of the information needs for which a software solution is desired
- Personnel — Software development team, managers, and supporting personnel
- Process — Practices, tools, and techniques employed by the personnel to develop the product; it proceeds in a series of steps (the software life cycle)
- Environment — Physical and informational resources and constraints within which the personnel and process operate
- Product — Software and documentation that solve the problem

Measures are needed to characterize the principal attributes of these components before the relationships among the components can be determined. A complete set of measures constitutes a profile. Software Development profiles form the baselines against which technologies are evaluated.

Evaluation

Even after assembling a substantial software engineering data base, other obstacles to accurate technology evaluation remain: technologies tend to be applied together, sample sizes are small, and many nontechnology factors also affect the outcome of a software development project. These complications prohibit a simplistic statistical analysis and interpretation of the software development measures collected (Reference 4). Nevertheless, some trends in the data are clear.

The SEL has extensively studied four factors: programmer performance, modern programming practices, quality assurance, and computer utilization. Figure 4 summarizes the results of an analysis of covariance performed with SEL data (Reference 7). Programmer performance proved to be the most important factor with respect to both productivity and reliability.

Figure 5 shows range of programmer productivity values encountered in the SEL data. The figure indicates that variantion is lessened (or performance is homogenized) in large projects. Figures 6, 7 and 8 plot data from 14 large attitude projects to illustrate the effects of the other factors

Modern programming practices

One group of individual technologies, referred to as modern programming practices, tend do be applied together. These techologies provide a flexible methodology for the (detailed) design, implementation, and verification of software.

As practiced in the flight dynamics environment, the principal components are as follows:
- Informal program design language
- Top-down development
- Structured programming
- Code reading
- Strutured FORTRAN preprocessor

The individual measures of technology use (listed above) were combined to form a single index of overall structured programming use. Figure 6 shows the relationship of this index to error rate. The use of modern programming practices appears to be associated with a reduced error rate. No significant correlation with productivity was found. This implies that the reliability benefits of modern programming practices are obtained at no additional cost in terms of development effort.

Quality assurance

Quality assurance includes all review and management procedures undertaken to ensure the delivery of an effective and realible product. The specific technologies studied by the SEL are as follows:
- Requirements Reviews
- Design Reviews
- Design Walkthroughs
- Code Walkthroughs
- Test Formalism
- Test Followthrough
- Methodology Reinforcement
- Document Quality Assurance
- Development Standards
- Code Configuration Control
- Code Library (PANVALET)
- Configuration Analysis Tool

For the analysis decribed here, the individual measures of technology use (listed previously) were combined to form a single index of overall quality assurance activity. Figure 7 shows the relationship of this index to error rate. Quality assurance activity appears to be associated with a reduced error rate. No significant correlation with productivity was found. This implies that the reliability benefits of quality assurance are obtained at no additional cost in terms of development effort.

Computer utilization

Another major factor in software development is the computing environment. Because changes to this environment usually cannot be made to conform to the needs of any single project, the computing environment is not considered to be a software engineering technology as defined in this paper. Nevertheless, it can have a strong effect on the development process (Reference 8). The flight dynamics computing environment provides the programmer with facilities for editing, compiling, linking, and testing source code.

Figure 8 shows that extensive computer use is associated with low productivity. Heavy computer users may not spend enough time desk checking and planning their work before jumping into code and test. However, a recent study (Reference 9) indicated that computer support for design and planning activities (not now provided) can increase the overall productivity and reliability of the software development process.

Transference

When the effectiveness of a technology has been demonstrated, the next step is to transfer it to software developers. The principal mechanisms used by the SEL to accomplish technology transfer include disseminating guidelines, developing tools, and conducting specialized training. The guidelines produced by the SEL cover management procedures (Reference 10), programming practices (Reference 11), and quality assurance (Reference 12). Two important SEL-developed tools are the Source Code Analyzer Program (Reference 13), which has been distributed across the United States, and the Configuration Analysis Tool (Reference 14), which is tailored to specific flight dynamics needs. Currently, SEL researchers are designing a training program for the Ada* language (Reference 15).

* Ada is a registered trademark of the U.S. Goverment, Ada Joint Program Office.

CONCLUSIONS

SEL experience demonstrated that the software development process can be improved by a throughful program of technology evaluation. Other similar organizations can also apply the lessons learned by the SEL. First, the use of modern programming practices increases software reliability without noticeably increasing development cost. Second, a regular program of quality assurance also improves software reliability at little or no net cost. Whereas many modern programming concepts are firmly established in software engineering practice, formal quality assurance procedures are only now coming into widespread use. Third, intensive computer use appears to be associated with low productivity. Programmers who spend a lot of time at the terminal tend to be less productive.

In summary, these results suggest that a formal and conscientious method of software development yields a more reliable product. On the other hand, it is very difficult to reduce the cost of *developing* a software product, although a more reliable product should require less subsequent maintenance. Despite technological advances, the major factor in both productivity and reliability continues to be personnel capability and performance (Reference 16).

ACKNOWLEDGMENT

The author would like to recognize the central roles of F. E. McGarry (National Aeronautics and Space Administration), G. T. Page (Computer Sciences Corporation), and V. R. Basili (University of Maryland) in planning and performing the studies described in this paper.

REFERENCES

1. B. A. Sheil, "The Psychological Study of Programming," ACM *Computing Surveys*, March 1981.
2. D. N. Card, F. E. McGarry, G. T. Page, et al., *Measuring and Evaluating Software Technology*, NASA/GSFC, under development.
3. D. N. Card, F. E. McGarry, G. T. Page, et al., *The Software Engineering Laboratory*, National Aeronautics and Space Administration/Goddard Space Flight Center (NASA/GSFC), SEL-81-104, February 1982.
4. B. W. Boehm, *Software Engineering Economics*. Englewood Cliffs: Prentice-Hall 1981.
5. T. DeMarco, *Controlling Software Projects*. New York: Yourdon Press, 1982.
6. V. E. Church, D. N. Card, F. E. McGarry, et al., *Guide to Data Collection*, NASA/GSFC, SEL-81-101, August 1982.
7. D. N. Card, F. E. McGarry, and G. T. Page, "Evaluating Software Engineering Technologies," *Proceedings of the Eighth Annual Software Engineering Workshop*, NASA/GSFC, SEL-83-007, November 1983.
8. F.E. McGarry, J. D. Valett, and D. L. Hall, "Measuring the Impact of Computer Resource Quality on the Software Development Process and Product," *Collected Software Engineering Papers: Volume 3*, NASA/GSFC, under development
9. K. Koerner, R. Mital, D. N. Card, and A. Maione, "An Evaluation of Programmer/Analyst Workstations," *Proceedings of the Ninth Annual Software Engineering Workshop*", NASA/GSFC, SEL-84-004, November 1984.
10. W. W. Agresti, F. E. McGarry, D. N. Card, et al., *Manager's Handbook for Software Development*, NASA/GSFC, SEL-81-205, April 1984.
11. W. J. Decker, *Programmer's Guide*, NASA/GSFC, under development.
12. Q. L. Jordan. *Product Assurance*, NASA/GSFC, under development.
13. W. J. Decker and W. A. Taylor, *FORTRAN Static Source Code Analyzer Program User's Guide*, NASA/GSFC, SEL-78-202, April 1985
14. W. J. Decker and W. A. Taylor, *Configuration Analysis Tool System Description and User's Guide*, NASA/GSFC, SEL-80-104, December 1982.
15. V. R. Basili, "Analysis of Software Development in Ada", *Proceedings of the Ninth Annual Software Engineering Workshop*, NASA/GSFC, SEL-84-004, November 1984.
16. F. E. McGarry, "Measuring Software Technology" *Proceedings of the Seventh Annual Software Engineering Workshop*, NASA/GSFC, SEL-82-007, December 1982.

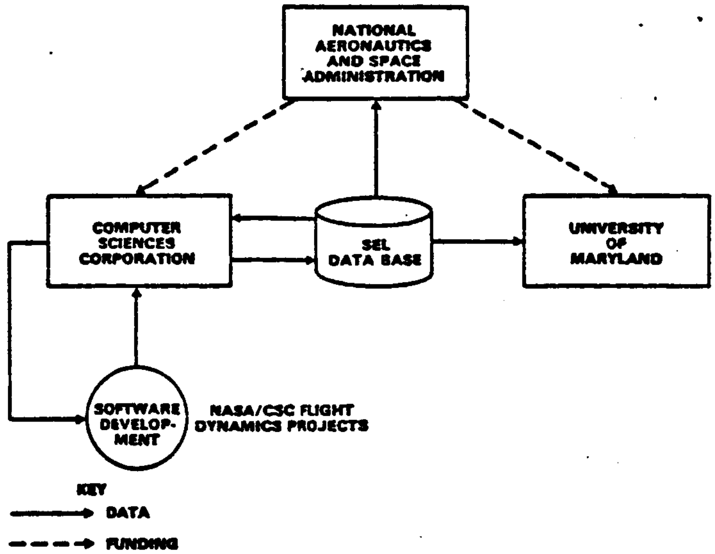


FIGURE 1 — Software Engineering Laboratory

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
INDIVIDUALS	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 2 — Flight Dynamics Software

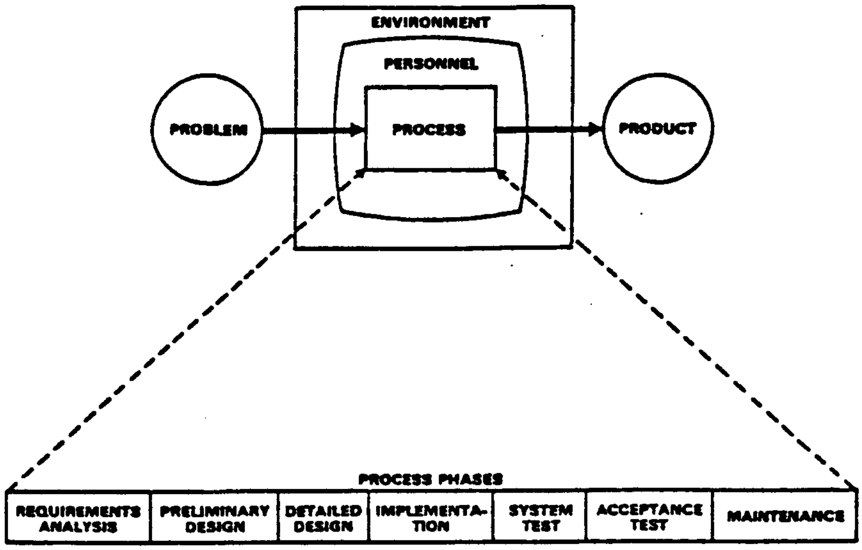
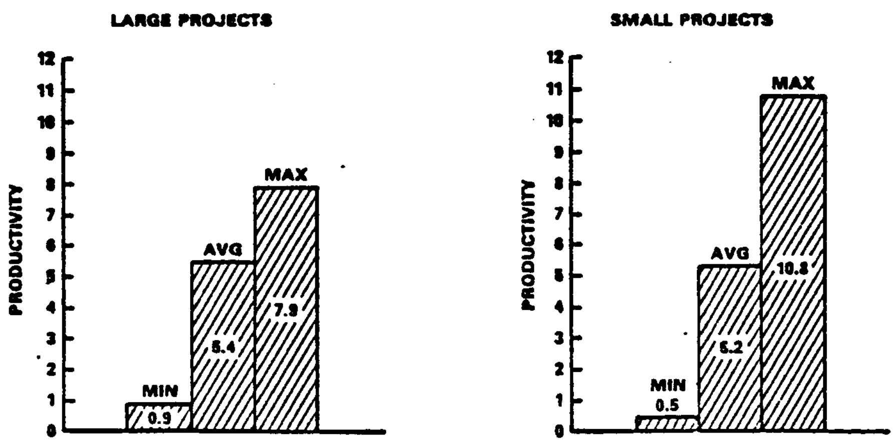


FIGURE 3 — Software Development Model

TECHNOLOGY FACTOR	PERCENT OF VARIANCE EXPLAINED BY FACTOR[a] PRODUCTIVITY	RELIABILITY
PROGRAMMER PERFORMANCE	28	46
MODERN PROGRAMMING PRACTICES	b	10
QUALITY ASSURANCE	b	10
COMPUTER UTILIZATION	28	b

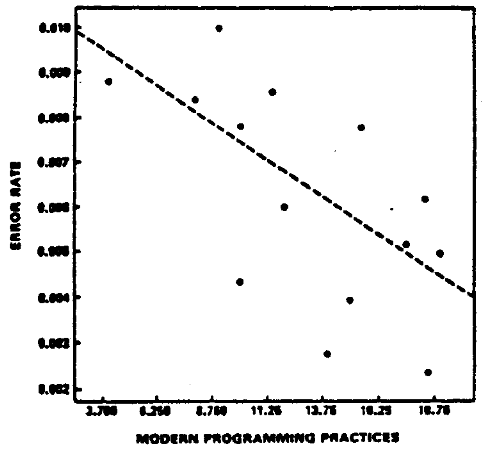
[a]VALUES FROM REFERENCE 7.
[b]NO SIGNIFICANT CONTRIBUTION.

FIGURE 4 — Technology Evaluations Summary



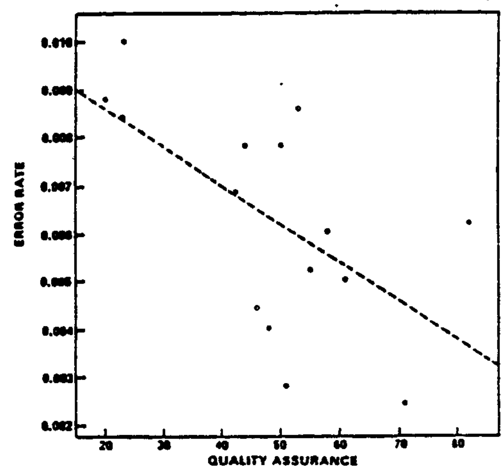
NOTES: LARGE PROJECTS ARE GREATER THAN 20,000 SOURCE LINES OF CODE.
PRODUCTIVITY IS SOURCE LINES PER STAFF HOUR.

FIGURE 5 — Programmer Productivity Variations



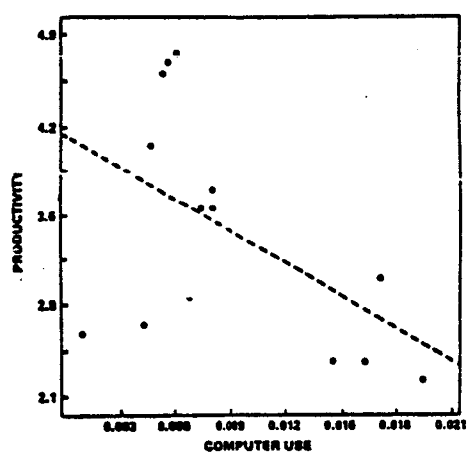
NOTES: ERROR RATE IS ERRORS PER DEVELOPED SOURCE LINE OF CODE.
MODERN PROGRAMMING PRACTICES IS INDEX COMBINING SIX MEASURES.

FIGURE 6 — Effect of Modern Programming Practices



NOTES: ERROR RATE IS ERRORS PER DEVELOPED SOURCE LINE OF CODE.
QUALITY ASSURANCE IS INDEX COMBINING 12 MEASURES.

FIGURE 7 — Effect of Quality Assurance



NOTES: PRODUCTIVITY IS DEVELOPED SOURCE LINES OF CODE PER HOUR.
COMPUTER USE IS COMPUTER HOURS PER DEVELOPED SOURCE LINE.

FIGURE 8 — Effect of Computer Use