UNCLASSIFIED

DTIC FILE COPY

(2)

REPORT DOCUMENTATION PAGE

AD-A198 398

2a. SECURITY CLASSIFICATION ---	1b. RESTRICTIVE MARKINGS
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S) AFOSR-TR- 88-0791
6a. NAME OF PERFORMING ORGANIZATION John Hopkins Univ.	6b. OFFICE SYMBOL (If applicable)
6c. ADDRESS (City, State and ZIP Code) Applied Physics Laboratory	7a. NAME OF MONITORING ORGANIZATION AFOSR
	7b. ADDRESS (City, State and ZIP Code) Building 410 Bolling AFB DC 20332-6448
8a. NAME OF FUNDING/SPONSORING ORGANIZATION AFOSR	8b. OFFICE SYMBOL (If applicable) NM
8c. ADDRESS (City, State and ZIP Code) Building 410 Bolling AFB DC 20332-6448	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER AFOSR 87-0219
	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO. 61102F; PROJECT NO. 2304; TASK NO.; WORK UNIT NO.
11. TITLE (Include Security Classification) Evaluation Methodology for Software Engineering (U)	
12. PERSONAL AUTHOR(S) Bruce I. Blum	
13a. TYPE OF REPORT Final	13b. TIME COVERED FROM 01 Jun 87 TO 31 May 88
14. DATE OF REPORT (Yr., Mo., Day) May 1988	15. PAGE COUNT
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES		
FIELD	GROUP	SUB. GR.

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The topic of this research involves two categories of investigation. One centers on the methods used for evaluation in the various scientific disciplines. The PI is studying these methods, but the research is not yet to the point that a unifying paper directed to the software engineering problem can be produced. The second area of investigation is that of the software process and what can be evaluated with respect to it. In this domain, work progresses through small experiments and conceptual studies.

Considerable accomplishments have been reported for the first year of research. There is every reason to believe that this progress will continue in the remaining two years of study and that some unified theory for process evaluation will evolve.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☒ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Abraham Waksman	22b. TELEPHONE NUMBER (Include Area Code) 202/767-5028
	22c. OFFICE SYMBOL NM

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

88 8 25 142

SECURITY CLASSIFICATION OF THIS PAGE

DTIC ELECTE AUG 26 1988 S E D

Evaluation Methodology for Software Engineering

A Report on the First Year of Research Under Grant No. AFOSR-87-0219

Bruce I. Blum

RMI-88-007
May, 1988

This work was supported by the Air Force Office of Scientific Research (AFOSR) under grant AFOSR-87-0219.

Evaluation Methodology for Software Engineering

A Report on the First Year of Research Under Grant No. AFOSR-87-0219

Bruce I. Blum

Johns Hopkins University/Applied Physics Laboratory

INTRODUCTION

This report describes the progress of the first year of research in a proposed three year program designed to establish the most effective methods for software engineering evaluation. The central concern is the impact of changes in the software process. In particular, there is a special interest in benefit improvement as demonstrated by evaluations across process models.

The research has pursued two types of activity. On the one hand, evaluation methods used in other disciplines have been reviewed for their utility in software engineering. The goal is to produce a taxonomy of methods with a suggested range of strengths for software engineers. The availability of this unified view would help analysts select the most appropriate evaluation techniques for a given class of task.

The second class of activity relates to small studies in which the evaluation methods can be tested and/or quantifiable concepts can be modeled. Because the research goal is to provide a means to appraise alternative development paradigms, some effort must be spent on the study of an essential software process model, i.e., a meta-process model.

This report is divided into two sections. The first describes the problem as it is interpreted in the context of this research grant. The second section presents the accomplishments of the first year of work.

THE TECHNICAL APPROACH

Computer science and the application of computers are undergoing revolutionary changes. Traditional development paradigms have new tools to support the software process. Examples include Ada and other languages that apply the principles of abstraction and concurrency management, environments and work stations that integrate graphics and text, and general purpose facilities that allow casual users to satisfy their needs directly. New paradigms also are being produced to offer improvements in quality, cost, and scope. Examples here are the use of artificial intelligence and knowledge-based assistants, the direct execution of specifications with the operational approach, and the application of new techniques such as object oriented programming and conceptual modeling. Finally, there are major changes in the hardware environment. For example, lowered costs eliminate many of the memory and processing speed barriers, new parallel architectures remove the earlier processing bottlenecks, and communications and networking blur the boundaries between

individual computers and databases.

Yet with all this improvement, we lack a clear understanding of how to evaluate our progress. In some areas, such as equipment cost per unit of memory or processing time per unit of operation, the change is easily quantified. However, when one sets as a goal the improvement of "the power, quality, reliability, and transportability of computer software and the verification of software, data, structure, and operating systems,"[1] how does one quantify the improvement? Moreover, if one asserts that the improvements result from the use of processes, methods, tools or environments, then how does one identify and evaluate the contributing factors?

This research addresses these issues. Methodology is the study of methods, and the focus of this investigation is the study of evaluation methods -- used in software engineering and in other scientific disciplines -- as they relate to software development, use and maintenance. The goal of this research is (a) to identify demonstrated techniques that can be applied in software engineering, (b) to establish taxonomies of (1) attributes that can be evaluated and (2) the associated evaluation methods, and (c) to document -- by means of references and pilot studies -- which metrics offer valid measures of improvement and which qualities can be evaluated only subjectively. Naturally, to provide a context for the evaluation, the research also involves a definition of the essential characteristics of the processes to be measured.

Problems in Software Engineering Evaluation

There are two classes of evaluation in software engineering. The first, which we call vertical evaluation, entails evaluation of software within a fixed context. The most common example of this type of evaluation is the use of measures of product size, changes and failures to evaluate cost, quality and reliability. Such evaluations normally are performed for a fixed development community with a given process model over an extended period of time. During the period of data collection there tend to be changes to the process and environment, and the data analysis is used either to measure improvements or to predict future performance. For example, cost models are based upon empirical evaluations of previously collected data. The vertical evaluations are most valid when they are based upon longitudinal data from a single organization. Comparisons across organizations have broader variance, and few industry-wide standards have been accepted.

Horizontal evaluations -- the target area of this research -- focus on the evaluation of technology with respect to its impact on the software process. That technology generally is presented in the form of a tangled hierarchy as follows:[2]

Process Descriptions
|
Methods and Practices
|
Tools
|
Support Environment

Thus, for a fixed process model and set of practices, there are tools that can support that model. There also are alternative approaches for combining these tools to

produce support environments for software development and maintenance. The goal of horizontal evaluation, therefore, is to measure the impact of changes at any of the four levels of this software engineering hierarchy.

Horizontal evaluation is difficult. First, computer science is unlike other sciences; its scientific base rests in the formalisms that it uses. These formalisms have logical properties that can be evaluated independent of any application. Indeed, Turski notes,[3]

> The history of advances in programming -- the little that there is of it-- is the history of the successful formalization: by inventing and studying formalism, by extracting rigorous procedures, we progressed from programming in machine code to programming in high level languages (HLLs).... For many application domains HLLs provide an acceptable linguistic level for program (and system) specification.... The expert views of such domains, their descriptive theories, can be easily expressed on the linguistic level of HLL, thus becoming prescriptive theories (specifications) for computer software.

In software engineering the primary object of interest is the software product, not its programming. Research is concerned with tools to support the development of descriptive theories in the problem domain, the transformations and practices necessary to formalize a HLL prescriptive theory that can be implemented as a software product, and the management and support of this process.

The kinds of evaluation appropriate for this research cannot follow the models of evaluation used in physics and engineering. There are no fixed phenomena; one cannot test a theory empirically because the data are affected by too many uncontrolled variables. This complexity also makes it difficult to separate the attributes to be evaluated from the background effects. The cost of collecting data is high, and there are difficulties in establishing controls in "real" (as opposed to "toy") projects. In fact, controlled studies with sample sizes large enough to evaluate a hypothesis are possible only for the most constrained issues.

In addition to the problem of not having well defined properties to be evaluated, there are -- as we noted in the discussion of vertical evaluation -- few broadly accepted baselines or "gold standards" against which one can measure change. Software engineering is dynamic, and it is not clear how data collected over a span of two decades can be used. For example, a frequently cited fact is that there can be a 1:28 variation in programmer performance. However, this is based upon one element in a 12 programmer study conducted in the late 1960s.[4] Is this valid in today's age of personal computers and computer literacy? Or was the difference an artifact of training that would correct itself as more effective methods were learned? Recall that the standard QWERTY keyboard format initially was chosen because it would slow performance and thus prevent the jamming of keys; this justification for its selection no longer is valid. In this spirit, some older assumptions about the software process should be reexamined.

Finally, there are inherent problems in the evaluation of software engineering technology. First, much of the research and development with respect to technology is either academic or proprietary. Evaluation in an operational environment may not be possible. Second, the academic research is complex and typically requires years to

complete. Thus, much that is reported must be descriptive, conceptual and/or subjective. Moreover, much of the software engineering technology that is investigated in a research setting has no parallel in a production setting. (For example, with the current levels of experience, it is not practical to evaluate the methods used to implement an expert system.) There also are unavoidable biases in evaluating a technology. Mahoney has studied the problem of self-deception in science and argues that "the psychological processes powerfully influence and, in many ways, constrain the quality of everyday scientific enquiry."[5] By way of conclusion, we note that all of these problems are further exacerbated when one performs this inquiry in a dynamic discipline, with a limited heritage of formal evaluation, and where there is a strong personal bonding with the objects of study.

A Framework for Evaluation

In establishing a framework for horizontal evaluation, there are two basic approaches. One can start by identifying the objects to be evaluated, or one could begin with methodological issues. We start with the first.

In a software engineering context, there are three objects that can be evaluated:

Problem. This is the application or need for which a software product is being developed.

Process. This is the sequence of activities associated with the software product's development and maintenance.

Product. This is the software item that is delivered and used.

There have been few attempts to fix a problem and investigate alternative approaches with respect to its implementation. Boehm's COCOMO system has been used as a control for some student exercises[6,7] and as a baseline for other studies.[8] Cugini has built a database of programs for a fixed, non-trivial problem, but the data have not been studied in any depth.[9] In each of these cases, a fixed problem was used to evaluate some properties of the process or the product. Halstead, on the other hand, introduced Software Science as a theory for repeatable and universal measures at the problem level.[10] The problems, in this case, were limited to algorithms, and much of the initial theory is no longer accepted as valid. Albrecht sought an alternative approach to quantifying the size of a problem; he introduced the concept of function points as a measure for information processing problems.[11] (Interestingly, the extensions to function point analysis focus on estimating the size of, and therefore the effort to produce, the end product.)

With respect to the process, there have been many evaluations of the impact of change within the context of a fixed process model. Most of this evaluation is what we already have termed vertical, i.e., it compares effects within a general problem domain and often within a single organization. The analysis of cost and schedule data are examples of this type of evaluation; such data would be useless for comparisons with process models that use tools or paradigms that distort the allocation of effort among the process steps. For example, how does one use historical costing data from traditional production cycles to estimate costs when using a Fourth Generation Language?

Finally, there is the software product. Most evaluations focus on attributes of the product.[12] Some obvious measures are lines of code and numbers of errors encountered. Code often is considered the first formal object that can be analyzed, and there are many easily computed metrics that are used to predict product quality. (McCabe's complexity metric[13] is one commonly applied example.) Nevertheless, virtually all evaluations of a final product hold the process model and environment fixed. Few studies are designed to address the impact of a major technological change, and many of those that do are naive in their study designs. For example, to what extent did the improvement associated with structured programming result from the introduction of discipline, the reduction in size of the conceptual objects being processed, or the Hawthorne effect? Although the question may seem facetious, if the structured approach was accepted because of its side effects, then its rigid retention may become a software parallel to the QWERTY keyboard.

Given this stratification of the problem domain, what evaluation methods can be applied? It is useful to start with a medical model.[14] At the lowest level, there is the basic research in biological phenomena. This involves *in vivo* and *in vitro* studies and the use of mathematical and animal models. The goal is to isolate some portion of a biomedical problem so that it can be understood better. Examples in the computer science domain include evaluation of algorithms and transformations, determination of user reaction responses to different interfaces, and the measurement of some properties of code or documentation. In each of these cases, the objective is to establish some invariants within a given context that add to our understanding of some larger problem.

At the next level are case studies and clinical trials. In medicine, more time is spent in training a specialist in a clinical setting than in a classroom. The amount of formal knowledge available is beyond the comprehension of a single individual; therefore, much of the physician's training is organized around the clinical situations that he is expected to encounter. The result is a set of learning experiences derived from case studies, i.e., specific instances. Experience has shown, however, that we are poor judges of outcome when we generalize from anecdotal records. Thus, medical research confirms its perceptions by the use of clinical trials. Here a cohort (a group of like patients) is selected, a set of procedures or therapies is defined that involves a limited number of variables, and the outcome is used to evaluate some null hypothesis. Most computer science examples of this type of evaluation are rooted in the behavioral sciences. The studies in individual differences among programmers are one example;[15] the recent workshops on the empirical study of programmers provide another illustration.[16]

The next higher level in the medical analogy is that of the health care delivery system. Epidemiology, for example, studies the health of the population and uses domain-specific knowledge to identify the causes of ill health, areas of potential risk, or the effects of change. The evaluation of a health care system, i.e., a system designed to alter the health status of a population, provides additional insights. In this case, the system considers benefits and costs separately. Costs are evaluated as dollar values. The benefits are organized into the following three categories:

Structure. This is the capacity of the facilities, qualification of the personnel, etc. An example in the software engineering context would be the use of methods and tools that achieve the goal of "requirements

analysis, design, test and maintenance of application software by technicians in an economics-driven context."[17]

Process. This is the volume, cost, and appropriateness of activities in the achievement of the system goals. A software example here would be the impact of walkthroughs as measured by the rates of defect detection in different stages of the development process.

Outcome. This is the change in status attributable to a system. Health-related examples would be mortality and morbidity rates. For software products, the measurable outcomes might be post-delivery error counts, evaluations of relative product performance or user satisfaction, and the ability to meet schedule or budget goals. Note that outcome measures always are relative to some baseline.

This statement of the research problem concludes with the following observation. Horizontal evaluation in software engineering just is emerging as a serious issue. There are many models to draw from in establishing evaluation methods, and the scientific quality of future research in software engineering will depend -- in part-- on how well we apply this knowledge. In the following section, the results of the first year's investigation are summarized.

YEAR 1 ACCOMPLISHMENTS

In the proposal for the first year of support, the research was divided into two categories of activity: conceptual and experimental. The goal of the conceptual tasks was to read, organize and document to gain an understanding of evaluation methodology as it relates to software engineering. The experimental tasks, on the other hand, involved trials in the software engineering domain that would provide insight into the methodological concepts.

One of the stated objectives was to produce a taxonomy of the objects, tools, methods, and environments to be evaluated plus a taxonomy of the scientific evaluation methods as they relate to software engineering. Considerable reading was done on the subject, and it became clear that additional reading would be required before such taxonomies could be presented for review. Thus, while the creation of the two taxonomies remains a research goal, the investigator does not feel that his current thoughts are mature enough to report on at this time.

The year one proposal identified a set of experimental tasks, each of which would require a man-month of effort to perform and evaluate. Six projects were identified, and it was stated that all would be initiated and three completed during year one. In late 1987 the author became the Principal Investigator of a research contract with the Office of Naval Research (ONR) to study knowledge representation in software engineering. This contract complemented the study under this AFOSR grant, and it was possible to enlarge several of the experiments and thereby consider both the evaluation methodology and knowledge representation issues.

The results of some of the research conducted under this grant have been documented in the form of reviewed papers, invited presentations, and internal reports. In addition, because the work was organized as a three year effort, there

are many projects in progress. In what follows, the published results are identified. The narrative presents each document in the context of the research activity, i.e., a search for problem clarification and potential solution. The detailing of the solution is contained in the referenced document; this report does not repeat the many findings already reported elsewhere.

The material is grouped into three categories: those activities that were supported exclusively by the AFOSR grant, those that were integrated with the research conducted for ONR, and those that were not directly supported by this grant. (The latter are included because they did have an impact on the PI's perceptions and thereby impacted his work in this grant.)

Activities Supported by AFOSR Grant Only

One of the topics of immediate interest involved the representation of the problem to be solved. That is, if a problem could be solved, and if it had a representation that implied the solution, then finding the qualities of that representation would provide the basis for the evaluation of alternative solutions and/or solution methods. Two short notes were prepared on different aspects of this problem. Each was presented at a meeting or workshop.

> Experience with an Atypical Development Environment Based Upon Reuse, Minnowbrook Workshop on Software Reuse, (Unpublished Workshop Notebook, 1987). To be reprinted in Will Tracz (ed.), *Software Reuse: The State of the Practice*, IEEE Computer Society Press. This short note introduced the concept of reuse as a formalization of prior experience. It then reviewed the forms that reusable objects might take. To provide a more concrete foundation for the discussion, the TEDIUM environment was used as a model to illustrate that such concepts are within the current state of the art.
>
> Experience with a Global Documentation Environment, *Second International Conference on Human-Computer Interaction*, 1987. This full page abstract addressed the idea of representation as knowledge about the application and then considered the issue of capturing that information in a global document environment. TEDIUM again was used to demonstrate that the concepts were sound. Evaluation of the environment in the context of human-computer interaction was the central concern of the presentation.

The author was invited to participate in a panel on software metrics at the IEEE Computer Standards Conference (COMPSTAN 88). The focus of this presentation was on the need to understand the process of evaluation and the danger of standardizing process metrics when the processes were subject to change. The talk was summarized in an abstract and an internal APL report. A draft paper that expands on this idea is being developed for the ACM SIGSOFT *Software Engineering Notes*.

> On the Use of Metrics to Compare Models, Methods and Tools, (abstract), *Proceedings, COMPSTAN 88*, IEEE Computer Society Press, 1988, p 107. Included in APL Research Center Report, RMI-88-006. This short note points out that evaluation is with respect to some baseline or other group, and that there are few well understood universals in software engineering

upon which to base an evaluation. Examples taken from recent issues of *Science* are used as illustrations.

Two refereed papers were produced. The first was prepared in response to an invitation by the editor of a special issue of *Large Scale Systems*. Of concern here was the use of new software development paradigms and their evaluation. The second was prepared for the 27th Annual Technical Symposium sponsored by the Washington, D.C. Chapter of the ACM. The topic for this symposium is Productivity: Progress, Perspectives, and Payoff. A paper was submitted and accepted that raised basic questions about what constituted productivity and how could it be measured.

Evaluating Alternative Paradigms: A Case Study, *Large Scale Systems*, (to be published in 1988). This is a full discussion of the software process and its evaluation. The major problems in development are identified along with generic solutions. Experience with TEDIUM is used to illustrate that the concepts are practical, and the paper concludes with a discussion of the risks associated with these new paradigms.

A Program a Day: Software Productivity's Four Minute Mile, *Proceedings, 27th Annual Technical Symposium*, (in press). This paper addresses the issue of productivity in the context of the differences between the problem and the product. The presentation draws upon Brooks' "No Magic Bullet" paper. It suggests alternative ways for looking at the problem that offer the potential for orders of magnitude improvements. Naturally, the question of how to evaluate the productivity growth also is considered.

Activities Supported by Both the AFOSR Grant and the ONR Contract

In late 1987 the author became a PI on an ONR contract to study knowledge representation issues in software engineering. Because there is a clear connection between the evaluation and software process concerns of this grant, the research effort was combined wherever a symbiotic benefit could be anticipated. Despite this occasional integration of research, some AFOSR tasks continue to be studied independently.

Three major papers were prepared as a joint research activity. The first was submitted to and accepted by the *Journal of Systems and Software*, the second was submitted to and accepted by the Conference on Software Maintenance, 1988, and the third was submitted to and accepted by the IFIP WG 8.1 Conference on Computerized Assistance During the Information Systems Life Cycle (CRIS 88). The last of these is being published in an extended form as an internal report. Finally, a short -- but conceptually important -- position statement was prepared for CASE '88.

Volume, Distance and Productivity, *Journal of Systems and Software*, (to be published in mid-1989). This paper introduces some concepts that provide insight into the software process and its productivity. The concepts, at this stage, are qualitative. Nevertheless, they get at the fundamental issues of representation and the evaluation of alternative paradigms (as they are realized in their representations). This has been a key paper in the structuring of my understanding of the development-productivity problem.

Documentation for Maintenance: A Hypertext Design, Conference on Software Maintenance, '88 (in press). Much of the discussion of hypertext deals with the technology; the presentation of an example is included to illustrate applicability. This paper considers the set of written text produced during the software process, i.e., the documentation, and proposes a model that can be managed using hypertext technology. The question of validating the model also is examined.

An Illustration of the Integrated Analysis, Design and Construction of an Information System with TEDIUM, *Proceedings of the IFIP WG 8.1 Conference on Computerized Assistance During the Information Life Cycle (CRIS 88)*, (in press). This conference solicited papers that used one of two sample problems to illustrate how automation could be applied to the information system life cycle. The author chose the more complex of the examples and implemented it. He then evaluated both the process and the product. Detailed notes were kept during the development activity, and insights were developed regarding the cognitive tasks during implementation, how they can be measured, and what kinds of tools could improve the process.

Closure and CASE Environments, (Position paper prepared for CASE '88). This brief paper helped to formalize the concept of closure in a software environment. If one considers the software process as a sequence of transformations, then clearly there is closure. However, when one projects the transformations onto the formal (i.e., automated) domain, the process is no longer closed; there are transformations from conceptual to formal representations. Human judgement is an essential factor in software development, and the concept of closure must be extended to the interface between these conceptual and formal representations. This paper explores some of these issues and suggests categories for measurement.

Activities Not Supported by the AFOSR Grant

In addition to the above activities, the author was involved in a variety of tasks that -- in various ways -- positively affected his work on this grant. None of these activities was directly supported by the AFOSR grant; however, attendance at some meetings may have been partially charged to the grant when it was felt that the topic was appropriate.

Two items of the work performed under the ONR contract were not involved with the central focus of the AFOSR grant, and the results of this effort have been published independently.

Human Information Processing in Software Development, *Concise Encyclopedia of Information Processing in Systems and Organizations*, Pergamon Press, (in press).

R.F. Wachter, A. Meyrowitz and _____, Expert Systems, Artificial Intelligence and Software Engineering, *Encyclopedia of Computer Science and Technology*, Marcel Dekker, Inc, (in press).

For some time the author has been involved with the field of medical informatics, and there have been some invitations to document his knowledge of that discipline. The following was accomplished during this period.

The ACM Conference on the History of Medical Informatics, (November, 1987). The author was the Conference Chair and is editing (with K. Duncan) a book on the conference. It will be published by the ACM Press and Addison Wesley.

The author is preparing several chapters for *A Clinical Information System for Oncology*, edited by J. Enterline, R. Lenhard and _____, Springer-Verlag. Much of this work involves issues in the evaluation of a medical system.

Medical Informatics, *Encyclopedia of Computer Science and Technology*, Marcel Dekker, (in press).

Computer Technology in the Health Sector in the Nineties, (invited presentation), Symposium Informatica Hospitaria, Barcelona, Spain, April, 1988.

The author also was a member of a project team that developed an Intelligent Navigational Assistant (INA) to serve as a natural interface to a large Army database. Some of the work was reported on during the life of this grant. The following paper was prepared during the grant period.

A Simple MUMPS Windowing Environment, 17th Annual MUMPS Users' Group Meeting, June, 1988. The author also will conduct a tutorial on Software Engineering at this conference.

Finally, we note that the author is active in the Johns Hopkins Continuing Professional Programs. During this grant period, he taught two different courses in software engineering and conducted a three day short course on the same topic. He also presented the following paper about software engineering education.

V. Sigillito, _____ and P. Loy, Software Engineering in the Johns Hopkins University Continuing Professional Programs, *Second SEI Conference on Software Engineering Education*, Springer-Verlag, (in press).

SUMMARY

The topic of this research involves two categories of investigation. One centers on the methods used for evaluation in the various scientific disciplines. The PI is studying these methods, but the research is not yet to the point that a unifying paper directed to the software engineering problem can be produced. The second area of investigation is that of the software process and what can be evaluated with respect to it. In this domain, work progresses through small experiments and conceptual studies.

Considerable accomplishments have been reported for the first year of research. There is every reason to believe that this progress will continue in the remaining two

years of study and that some unified theory for process evaluation will evolve.

REFERENCES

1. Mathematics and Information Sciences, *Research Interests of the AFOSR*, Bolling Air Force Base, Washington, D.C. 20332, November, 1985, p. 37.

2. Musa, J. (ed.), Stimulating Software Engineering Progress, A Report of the Software Engineering Planning Group, *ACM SIGSOFT SEN*, (8,2):29-54, 1983.

3. Turski, W.M., The Role of Logic in Software Engineering, *Proceedings, 8th International Conference on Software Engineering*, IEEE Computer Society Press, 1985, p 400.

4. Sackman, H., et al., Exploratory Experimental Studies Comparing Online and Offline Programming Performance, *Communications of the ACM*, (11,1), 1968.

5. Mahoney, M.J., Self-Deception in Science, *AAAS Annual Meeting*, 1986, draft preprint, p. 1.

6. Boehm, B.W., An Experiment in Small-Scale Application Software Engineering, *IEEE Transactions on Software Engineering*, SE-7:482-493, 1981.

7. Boehm, B.W., T.E. Gray and T. Seawaldt, Prototyping vs. Specifying: A Multi-Project Experiment, *IEEE Transactions on Software Engineering*, SE-10:290-303, 1984.

8. Blum, B.I., A Paradigm for Developing Information Systems, *IEEE Transactions on Software Engineering*, SE-13:432-439, 1987.

9. Cugini, J.V., Selection and Use of General Purpose Programming Languages (2 Vols.), NBS Spec Pub 500-117, 1984.

10. Halstead, M. *Elements of Software Science*, Elsevier, Amsterdam, 1977.

11. Albrecht, A.J. and J.E. Gaffney, Jr., Software Function, Software Lines of Code, and Development Effort Predictions: A Software Science Validation, *IEEE Transactions on Software Engineering*, SE-8:629-648, 1983.

12. Basili, V.R., R.E. Selby and D.H. Hutchens, Experimentation in Software Engineering, *IEEE Transactions on Software Engineering*, SE-12:737-743, 1986.

13. McCabe, T., A Complexity Measure, *IEEE Transactions on Software Engineering*, SE-2:308-320, 1976.

14. Blum, B.I., *Clinical Information Systems*, Springer-Verlag, New York, NY, 1986.

15. Curtis, B., Fifteen Years of Psychology and Software Engineering: Individual Differences and Cognitive Science, *Proceedings, 7th International*

Conference on Software Engineering, IEEE Computer Society Press, pp. 97-106, 1984.

16. *Empirical Studies of Programmers*, Ablex Publication, Corp., Norwood, NJ, 1986, 1987.

17. Boehm, B.W., Software Engineering, *IEEE Transactions on Computers*, C-25:1226-1241, 1976, p. 1239.