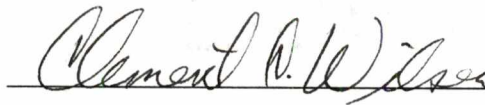
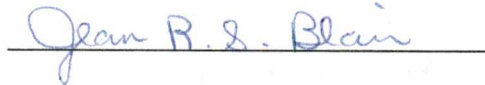


To the Graduate Council:

I am submitting herewith a thesis written by Carmen J. Trammell entitled "An Investigation of the Reliability and Validity of a Group Process for Defining Local Software Quality." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.


J. H. Poore, Major Professor

We have read this thesis
and recommend its acceptance:



Accepted for the Council:


Vice Provost
and Dean of The Graduate School

Statement of Permission to Use

In presenting this thesis in partial fulfillment of the requirements for a Master's Degree at the University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under the rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgement of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of InterLibrary Services when, in the opinion of either, the proposed material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature Carmen Skennell

Date April 24, 1991

AN INVESTIGATION OF
THE RELIABILITY AND VALIDITY OF A GROUP PROCESS
FOR DEFINING LOCAL SOFTWARE QUALITY

Carmen J. Trammell

May 1991

A Thesis Presented for
the Master of Science Degree in Computer Science
The University of Tennessee, Knoxville

ACKNOWLEDGEMENTS

The work reported in this thesis follows previous work on local measures of software quality by Jesse H. Poore, Chair of the Department of Computer Science at the University of Tennessee, and Leon H. Binder, Director of the Data Systems Department at Maryville College in Maryville, Tennessee. I have enjoyed participating in the stream of research they began, which aspires to contribute to not only the state of the science but the state of the practice.

I appreciate the advice and participation of the members of my thesis committee: Dr. Poore, chair; Dr. Jean Blair, Department of Computer Science; and Dr. Clement Wilson, Department of Mechanical Engineering.

Several fellow students in the Computer Science Department participated in a pilot application of the group process for defining local software quality described in this thesis. Dan Fetzer, David Fuhrer, Mohamad El Jazzar, and Hailong Mao were extremely helpful in working with me to prepare for field applications of the process.

The Information Systems Department of the Knoxville Utilities Board (Knoxville, Tennessee) and the Office of Scientific and Technical Information group with Science Applications International Corporation (Oak Ridge, Tennessee) were the organizations in which field applications of the group process were conducted. A debt of gratitude is due the managers who facilitated our involvement and the practitioners who participated in the process as "jurors."

Several other organizations supported the research by allowing individual employees to participate in research tasks. My sincere thanks to the sixty-eight people from Computer Technology and Imaging, Inc; Plasti-Line, Inc.; the Manufacturing Applications Department of Martin Marietta Energy Systems; the University of Tennessee Office of Information Systems; and the University of Tennessee Computer Science Department who served as research "subjects."

ABSTRACT

"Global" measures of software quality are generally not used by practitioners because they have not been calibrated for local operating environments. In a University of Tennessee research program on local measures of software quality, a participative process for defining and measuring local software quality is being refined and tested in field applications. The thesis of this research is that measures of software quality are most valid and most credible when they are based on local experience with the particular parameters that define the local computing environment.

The group process for defining local software quality involves having a "jury" of seasoned practitioners evaluate a sample of modules from the organization's inventory of code. Good practices are distinguished from bad practices as modules are rank ordered through a structured group process. One of the outcomes of the process is a "software quality rule set" that reflects the best practices currently in use in the organization.

In this study, the results of four field applications of the process were tested in formal validation experiments to determine whether the obtained rule set is a valid representation of the organization's sense of quality. Nonorganizational programmers were asked to rank the same set of modules that were ranked by the organization's jury, using either the locally-derived rules, generic (textbook) rules, placebo (nonhelpful) rules, or no rules. In three of four tests, programmers using the local rule set ranked the modules in a way that correlated more strongly with the jury's ranking than any other group. The results of the fourth test would have been consistent with the others but for a single individual's results. The overall correlations between the test groups and juries were .579 for local rules, .479 for generic rules, .446 for placebo rules, and .483 for no rules. The correlation between the local rules group and the jury was statistically significant at $p < .0005$; all other correlations were significant at $p < .005$. The results support a conclusion that the group process for defining local software quality is reliable in producing valid rule sets.

TABLE OF CONTENTS

Chapter	Page
I. Introduction.....	1
Participation and Quality in American Industry.....	1
Participation and Quality in the Software Industry.....	2
Industrial Usage of Software Quality Metrics.....	5
Local Derivation of Software Quality Metrics.....	7
Statement of the Problem.....	8
Definition of Terms.....	8
Research Plan.....	9
Limitations of the Study.....	10
II. Review of the Literature.....	12
Group Decision-Making Processes.....	12
Decision-Making Strategies.....	12
Decision-Making Techniques.....	13
Techniques Used by Quality Circles.....	13
Other Group Decision-Making Techniques.....	15
Comparative Research on Group Decision-Making Techniques.....	17
Conclusions	20
"Rules" for Software Quality.....	20
Textbook Rules.....	20
The Role of Aesthetics.....	21
Research on Rules.....	21
Conclusions.....	23
III. Research Procedures.....	24
Overview of Method.....	24
Process for Derivation of Rule Sets.....	25
Experimental Design for Testing of Rule Sets.....	37
Subjects.....	38
Data Analysis.....	39

IV. Results.....	40
Test of Validity for OIS Rule Set.....	41
Test of Validity for CTI Rule Set.....	45
Reliability of Process Used by Binder.....	49
Test of Validity for SAIC Rule Set.....	50
Test of Validity for KUB Rule Set.....	54
Reliability of New Process For Deriving Rule Sets.....	58
Reliability and Validity Across Tests.....	59
Reliability and Construct Validity.....	59
Reliability and Criterion Validity.....	59
Aggregate Validity.....	60
Aggregate Reliability.....	62
Key Findings.....	63
 IV. Summary and Conclusions.....	 64
Reflections on Field Applications of the Group Process.....	65
Sources of Variability in Experimental Use of Rule Sets.....	68
Using Rule Sets in Practice.....	71
Continuation of This Stream of Research.....	72
 Bibliography.....	 74
 Appendices	
A. OIS Rule Set.....	79
B. CTI Rule Set.....	80
C. Sample Composite Preliminary Evaluation.....	81
D. Rules from <i>The Elements of Programming Style</i> by Kernighan and Plauger.....	92
E. Placebo Rule Set.....	94
F. SAIC Rule Set.....	95
G. KUB Rule Set.....	98
 Vita.....	 100

LIST OF TABLES

Table	Page
1. Nippon Electric Software Quality Circle Achievements.....	4
2. Rank Ordering of OIS Modules: Raw Data.....	42
3. OIS Rule Set Test: Individual Correlations for Local Rules Group.....	43
4. OIS Rule Set Test: Individual Correlations for Generic Rules Group.....	43
5. OIS Rule Set Test: Individual Correlations for Placebo Rules Group.....	44
6. OIS Rule Set Test: Individual Correlations for No Rules Group.....	44
7. OIS Rule Set Test: Correlations Between Test Groups and Jury.....	45
8. Rank Ordering of CTI Modules: Raw Data.....	46
9. CTI Rule Set Test: Individual Correlations for Local Rules Group.....	47
10. CTI Rule Set Test: Individual Correlations for Generic Rules Group.....	47
11. CTI Rule Set Test: Individual Correlations for Placebo Rules Group.....	48
12. CTI Rule Set Test: Individual Correlations for No Rules Group.....	48
13. CTI Rule Set Test: Correlations Between Test Groups and Jury.....	49
14. Rank Ordering of SAIC Modules: Raw Data.....	51
15. SAIC Rule Set Test: Individual Correlations for Local Rules Group.....	52
16. SAIC Rule Set Test: Individual Correlations for Generic Rules Group.....	52
17. SAIC Rule Set Test: Individual Correlations for Placebo Rules Group.....	53
18. SAIC Rule Set Test: Individual Correlations for No Rules Group.....	53
19. SAIC Rule Set Test: Correlations Between Test Groups and Jury.....	54
20. Rank Ordering of KUB Modules: Raw Data.....	55
21. KUB Rule Set Test: Individual Correlations for Local Rules Group.....	56
22. KUB Rule Set Test: Individual Correlations for Generic Rules Group.....	56
23. KUB Rule Set Test: Individual Correlations for Placebo Rules Group.....	57

24.	KUB Rule Set Test: Individual Correlations for No Rules Group.....	57
25.	KUB Rule Set Test: Correlations Between Test Groups and Jury.....	58
26.	Average Individual Correlation with the Jury for All Subjects in All Groups.....	61
27.	Average Group Correlation with the Jury for All Groups in All Tests.....	61
28.	Average Intercorrelation Between Subjects.....	63
29.	Summary of Correlations Between Test Groups and Juries.....	69

LIST OF FIGURES

Figure	Page
1. Decision Strategies as a Function of Group Members' Perspectives.....	12
2. Accuracy of Group Decisions as a Function of Interaction Pattern.....	19
3. Preliminary Evaluation Form.....	29
4. Individual Module Ranks Form.....	32
5. Agreement Pattern Form.....	33
6. Posttest Only Control Group Design.....	39
7. Individual Correlations with Juries.....	60
8. Intercorrelations Between Subjects in the Same Test Groups.....	62

CHAPTER I

INTRODUCTION

Computer software has historically been exempt from the product quality expectations that have existed for other types of products and services. Neither old industrial approaches to quality (product inspection by quality control personnel) nor new approaches (process improvement through worker participation) have been widely applied to software development.

Growing concern about reliability of software has, however, resulted in increased interest in methods of assessing and improving the quality of software. A number of global metrics for software quality have been proposed, but little attention has been given to practical adaptation of these metrics to local software development environments. Less attention still has been given to methods of quality improvement that enlist local practitioners in the setting of their own quality standards.

A participative approach to local derivation of software quality measures has been proposed by Poore (1988) and tested in several field experiments by Binder and Poore (Binder, 1987, 1988a, 1988b, 1988c; Binder and Poore, 1990). The present study represents a continuation of this stream of research.

Participation and Quality in American Industry

A revolution in American management practices began in the late 1970's as a reaction to what was commonly referred to as the "productivity crisis." Faced with a decline in U.S. market share in virtually every industry worldwide, corporate America attributed its

problems to aggressive unions, overregulation by the federal government, unfair trade practices, fluctuation in the dollar, and a host of other factors before examining its own role in the loss of worldwide competitiveness.

By the early 1980's, lessons from the Japanese (Ouchi, 1982) and "the best" of American industry (Peters and Waterman, 1982; Levering, 1984) had begun to be understood: successful companies in both the U.S. and Japan put management emphasis on the interdependent objectives of quality improvement and participative management. The interdependence of these objectives was manifest in Japan by the widespread existence of "quality circles," small groups of employees from a given work area that meet on an ongoing basis to solve the company's problems.

Although American industry had some initial difficulty in importing quality circles and other management techniques that have been key to Japanese industrial success, over 90% of Fortune 500 companies have quality circles today.¹ The basic lesson of "quality through participation" has been heard: *people are most committed to standards of quality that they set for themselves.*

Participation and Quality in the Software Industry

Few reports of efforts to effect "quality through participation" in software development organizations have appeared in the literature. Of those that have appeared, the Japanese experience has been notable for its breadth.

Kellogg (1986) has described the Japanese "software factory," a framework in which software is designed, implemented, and maintained. Quality circles are at the heart of the software factory and of Japan's Project SIGMA for the industrialization of software in

¹ E.E. Lawler and S.A. Mohrman, "Quality Circles After the Fad," *Harvard Business Review*, January-February 1985, p. 66.

Japan. Announced in 1985 by the Ministry of International Trade and Industry, Project SIGMA has as one of its goals a fivefold increase in Japanese software production capability. This goal is particularly striking in light of an analysis of Japanese software factories by McNamara in which existing software production capability was estimated as six times greater than equivalent U.S. efforts.² Kellogg comments that "Domestic software industries within the United States...must either match the efficiency of Project SIGMA or cede the marketplace to it." ³

Kellogg's appraisal that the five-year timetable for Project SIGMA is realistic was based in part on his personal observations of the Fujitsu and Nippon Electric Company (NEC) software factories and their quality circles. Mizuno (1983) of NEC has written in more detail about the company's Software Quality Control (SWQC) program. SWQC is a campaign for quality management in all software-related work. The goals of SWQC are high quality software and high software productivity. SWQC is carried out by groups called Quality Control (QC) Circles. Mizuno described the individual and team benefits of QC Circles, and gave the results in Table 1 as examples of circle achievements.

A few American companies' experience with participative approaches to software quality have also been reported.

Ryan (1986) described the "Software Quality Circle" within Texas Instruments' Computer Systems Division, which is composed of representatives of each development group in the division. The Software Quality Circle works in concert with the Software Quality Branch, which is a department within the Computer Systems Division responsible for administering the Division's development methodology. The Circle formulates

² D.L. Kellogg, "The Implications of Japanese Software Factories and Project SIGMA Within Software Development Environments," Proceedings of the IEEE/AIAA 7th Digital Avionics Systems Conference, 1986, p. 29.

³ Ibid., p. 31.

Table 1. Nippon Electric Software Quality Circle Achievements ⁴

<i>Division</i>	<i>Target</i>	<i>Results</i>
Switching	Machine Time for Debug	Down by 1/3
Transmission Control	Bug Ratio	1.37/KS --> 0.41/KS
Minicomputer	Bug Ratio	0.35/KS --> 0.20/KS
Large OS	Bug Ratio	6/month --> 0.9/month
	Source Size	20KS --> 8KS
	Object Size	72KB --> 26KB
Large APs	Spec Changes	Down 40%

standards and procedures, acting in an advisory capacity to the Software Quality Branch.

Binder (1988b) reported a participative approach to the development of programming standards in the Engineering Management Information Systems (EMIS) section of the Computing and Telecommunications Division of Martin Marietta Energy Systems.

In 1985, the EMIS section head appointed a committee of the three newest employees to develop a tentative set of programming standards to be discussed and finalized by the entire EMIS team. The establishment of a workable set of standards required many meetings and evolved over a period of several months. There were many heated discussions, often with lines drawn between the least and most experienced employees. In a few cases, the standards were arbitrarily set by the EMIS section head or remained too loosely defined due to a general lack of agreement among team members. Standards were first reluctantly accepted and peer reviews were unevenly conducted. Acceptance came gradually over a period of approximately six months when team members realized they each had a voice in the setting of the standards and when they perceived significant improvement in the quality of their jobs. With standardization, reading

⁴ Y. Mizuno, "Software Quality Improvement," *Computer*, March 1983, p. 71.

other team members' code was less tedious. Maintenance became easier and less time-consuming. Soon team members began having serious discussions on ways to improve the standards; programming standards were upgraded by agreement of team members continually for the first two years...a permanent EMIS Standards Committee was established ...the success of standardization is remarkable...EMIS team members spend less than fifteen percent of their efforts on system maintenance...All EMIS team members emphatically agree that programming standardization is essential for computing efficiency and professionalism.⁵

Phillips (1986) has described the use of software quality teams in the software product development environment at IBM. The members are product developers, a product designer who is the leader/facilitator, and the programming process group. The team members are volunteers who spend approximately 25% of their time on quality team activity while continuing with their regular duties. A management steering committee participates once a month for the purpose of monitoring team progress and providing the team with necessary resources. Phillips described one team's focus on "causal analysis and defect extinction" in software programs for which they are responsible. As a result of team actions, a 50% reduction in pre-ship defects has been experienced, as well as a 50% reduction in post-ship defects.⁶

Briefer accounts of quality circle activities at Honeywell (Pickler, 1983), Boeing (Couger, 1983), and Bell Laboratories (Prell, 1984) have also been reported.

Industrial Usage of Software Quality Metrics

A major focus of research in software quality to date has been on the development of metrics that characterize either the composition or performance of software. Most of these

⁵ L.H. Binder, "Local Quality Metric for CTD COBOL Programs," University of Tennessee Technical Report, April 1988, pp. 2-4.

⁶ R.T. Phillips, "An Approach to Software Causal Analysis and Defect Extinction," GLOBECOM '86: IEEE Global Telecommunications Conference Record (New York: IEEE, 1986), pp. 415.

global metrics "...suffer from lack of field testing and calibration," and as such, lack predictability and credence in specific software environments (i.e., with particular hardware, operating system software, and programming languages).⁷

Global metrics have generally not been embraced by practitioners. Some companies, however, have attempted to combine global metrics with metrics of their own.

Texas Instruments uses "leading, concurrent, and lagging quality indices."⁸ Halstead's Effort Metric is an example of TI's leading quality indices, which are used in the development phase of a project. Concurrent quality indices reflect experience during testing, such as frequency and severity of "Software Trouble Reports." Lagging quality indices reflect Software Trouble Reports after release.

Prell and Sheng (1984) reported that Bell Laboratories uses both standard and internally developed metrics. Examples of standard quality metrics used are fault density, fault discovery and correction rates, and number of open problems and their severities. In addition, metrics that reflect performance of a system as perceived by the user have been developed. Examples of such metrics for a switching system are capacity to switch phone calls, reliability to provide continuing service, percentage of calls completed, and frequency and duration of recovery actions. These metrics are major considerations in assessing overall system performance.

Grady and Caswell (1987) described a company-wide metrics program at Hewlett-Packard in which a corporate Software Metrics Council arrived at a standard set of metrics through a "bottom-up" development process. HP has a tradition of local autonomy in its organizational processes, encouraging innovation through divisional experimentation.

⁷ J.H. Poore, "Derivation of Local Software Quality Metrics," *Software---Practice and Experience*, November 1988, p. 1017.

⁸ J.R. Ryan, "Software Product Quality Assurance," AFIPS Conference Proceedings (Arlington, Va.: AFIPS Press, 1982), p. 398.

Lessons learned from divisional approaches were reviewed by the Council, and metrics that had proven most useful in contributing to process improvement were included in the corporate set, which was disseminated throughout the company.

Local Derivation of Software Quality Metrics

In "Deriving Local Software Quality Metrics" (1988), Poore proposed a process by which the software development staff in a given organization could derive and use their own measures of software quality. The steps in the process are as follows.

1. An initial concept of quality for the local environment is articulated by a "jury of experts" from a given software organization.
2. The jury ranks a randomly selected set of modules from the organization's software inventory in order of perceived quality, stating the rationale for each ranking decision.
3. The stated rationale for ordering of modules is summarized in a "rule set" that reflects the group's operational sense of quality.
4. Statistical analysis of the syntax of the modules is performed to search for a metric that produces a ranking of modules which correlates highly with the jury's ranking.
5. The statistical metric (if found) can be used for ongoing evaluation of the organization's software inventory, and the rule set can be used to guide programmers in improving the quality of old code and ensuring the quality of new code.

Binder (1987, 1988a, 1988b, 1988c) performed four experiments based on Poore's method. In each experiment, a statistical metric was found that produced a ranking of modules that correlated highly with the jury's ranking. A rule set was also produced in each experiment, but was not tested as an independent predictor of module ranking.

The reliability of the process for deriving a valid metric was established in the Binder experiments and reported by Binder and Poore (1990). The validity of the rule set and the reliability of the process by which it is derived have not yet been investigated, and were the subject of this research.

Statement of the Problem

The purpose of this study was to investigate the reliability and validity of a process by which a jury of local experts derives a rule set that specifies programming practices associated with high quality software in the local environment. The construct and criterion validity of the rule sets were examined, and the reliability of the process in producing valid rule sets was investigated. Specifically, the research questions were as follows.

1. Is a "locally-derived" rule set any more effective in enabling "naive programmers" to understand a work group's sense of software quality than a "generic" rule set, a "placebo" rule set, or personal judgment alone? (This is the test of construct validity.)
2. Does a locally-derived rule set enable naive programmers to make judgments about local software that correlate significantly with the judgments of local experts? (This is the test of criterion validity.)
3. Are valid results consistently obtained in successive uses of the process? (This is the test of reliability.)

Definition of Terms

The following terms are operationally defined for use in this research.

- Expert - A person who is a member of a jury.
- Expert judgment - The professional judgment of a member of a jury of experts.
- Generic rule set - The programming rules given in *The Elements of Programming*

Style by Kernighan and Plauger.

- **Jury** - A small group of employees selected by the head of a programming organization for their "representativeness" of the staff who meet to rank a set of modules in order of quality and state their rationale for each ranking decision.
- **Locally-derived rule set** - A rule set that reflects the judgment of local experts about requirements for software quality in their organization; a set of prescriptive statements made by experts to explain judgments about software quality.
- **Module** - A segment of code (delimited by function) that has been randomly selected from an organization's inventory of software; generally a subprogram.
- **Naive programmers** - persons who (1) are not employed by the organization which derived the rule set being tested, (2) are either upperclass undergraduate students with at least one academic year of coursework in computer science, graduate students in computer science, or practitioners with at least two years of programming experience, and (3) say that they have a good working knowledge of the programming language relevant to the experiment.
- **Placebo rule set** - A rule set intended to be of no help in evaluating the quality of a set of modules.
- **Rule** - A prescriptive statement.
- **Rule set** - A collection of rules.
- **Software quality** - the degree to which source code reflects the programming practices that are considered important in a given setting.

Research Plan

The investigation of validity was conducted under a posttest only control group design. Four groups of naive programmers used different approaches to make judgments about the quality of a target organization's software. The four approaches involved use of

(1) the locally-derived rule set (i.e., the rule set derived in the organization whose software was under evaluation), (2) a generic programming practices rule set, (3) a placebo (i.e., nonhelpful) rule set, and (4) personal judgment alone. The effectiveness of the approaches was compared to determine which best enabled subjects to make judgments that correlated with those of the target organization's experts.

Four such investigations were conducted, two involving rule sets developed in Binder's work and two involving newly-developed rule sets. The consistency of findings across investigations was examined to determine the reliability of the process used.

Limitations of the Study

External Validity. Poore stresses the importance of fixing as many parameters related to operating conditions as possible (e.g., organizational culture, application, hardware, operating system software, programming language). Findings are not intended for generalization beyond the environment in which they were derived.

Threats to Internal Validity. Applied research is invariably fraught with confounding variables and unintended sources of bias. In the case of the proposed study, where an interactive group process will be used to make judgments by consensus, a number of factors that relate to group dynamics may have unintended effects on the process. Examples of such factors are as follows.

- *Group Maturity/Group Norms.* It has been assumed that members of a working group can efficiently reach consensus about the relative quality of modules in a sample because, to some degree, they already have a "working local notion of quality." ⁹ Group norms do not develop overnight. Social processes define group norms. Time and interaction are required for group maturity and the development of norms. No test of group maturity will be done in this research to determine whether the existence of group norms, i.e., "local notions," are likely to exist.

- *Group cohesiveness/"Groupthink"*. Excessively cohesive groups often exhibit a phenomenon that has been called "groupthink," in which conflict and disagreement that could challenge the harmony of the group are suppressed (Janis, 1972). The hallmark of such groups is poor decisions, because an insufficiently critical approach to consideration of alternative points of view is taken. No assessment of group cohesiveness will be done in this research to determine the potential impact of the groupthink phenomenon on decision quality.
- *Individual Conflict of Interest*. A number of sources of individual bias are conceivable. Jury members who are also authors of the software under review can hardly be considered objective evaluators. Jury members who know they are reviewing the work of an author who is present are likely to curb critical commentary. Jury members who are subordinate in organizational status to other jury members may have some inhibition about honest commentary. None of these issues will be addressed in this research.

The aforementioned factors are only a few of the many social variables over which the real world rarely affords complete control. The proposed research was conducted with "reasonable prudence" with respect to these variables, if not perfect experimental control.

⁷ J.H. Poore, "Derivation of Local Software Quality Metrics," *Software---Practice and Experience*, November 1988, p. 1017.

CHAPTER II

REVIEW OF THE LITERATURE

This study involves the use of a group decision-making process to develop a "rule set" that reflects the working sense of software quality for a given organization. There are two areas of literature, therefore, that are relevant to the study: group decision-making processes, and existing "rules" (i.e., standards or guidelines) concerning software quality.

Group Decision-Making Processes

A theoretical framework for group decision making is presented as a basis for selecting a decision-making strategy appropriate to the proposed study. Following the discussion of strategy, relevant research on group decision-making techniques is reviewed.

Decision-Making Strategies. Thompson and Tuden (1959) described two factors that determine the strategy appropriate for group decision making: agreement/lack of agreement on goals, certainty/uncertainty about the effects that various courses of action might have on the attainment of the goal. The four strategies that are produced by various combinations of these factors are shown in Figure 1.

		GOALS	
		Agreement	Disagreement
BELIEFS ABOUT CAUSE AND EFFECT	Certainty	<i>Computation</i>	<i>Compromise</i>
	Uncertainty	<i>Judgment</i>	<i>Inspiration</i>

Figure 1. Decision Strategies as a Function of Group Members' Perspectives

¹⁰ J. Thompson and A. Tuden, "Strategies, Structures, and Processes of Organizational Decision Making," in *Comparative Studies in Administration* (Pittsburgh: University of Pittsburgh Press, 1959), p. 198.

Characteristics of these strategies are as follows.

- Computation - quantifiable, systematic, programmable
- Compromise - pragmatic, bargaining
- Judgment - experience-based, consensual
- Inspiration - intuitive, creative, emotional

According to this view of decision making, the proposed study might most appropriately employ the Judgment strategy for group decision making since previous work in this stream of research has shown that (1) there is likely to be agreement on the goals of the proposed group activity (i.e., to establish a basis for improving the overall quality of the organization's software inventory), and (2) there is likely to be some disagreement over the effect of various programming practices on the quality of software.

Decision-Making Techniques. The following review of group decision-making techniques is covered in three sections: techniques commonly used in quality circles throughout industry, well-established techniques that are commonly used in a variety of settings (but not "institutionalized" for any particular purpose), and comparative research on these techniques.

Techniques Used By Quality Circles

A quality circle is a small group of employees who meet regularly to identify, analyze, and solve a company's problems. Quality circles originated in Japan in 1962 as part of the larger effort by Japanese industry to improve the quality of its products.¹¹

The first U.S. quality circle program was established in 1974 at Lockheed Missile

¹¹ F.M. Gryna, Quality Circles: A Team Approach to Problem Solving (New York: AMACOM, 1981), p. 9.

and Space Division in Sunnyvale, California. Now widely used throughout industry, quality circles are thought to provide a return on investment of 6 to 1.¹²

The process and techniques that are standard in quality circles are listed as follows (Gryna, 1981; Couger 1983).

PROBLEM SELECTION

- Listing
- Prioritizing

DATA COLLECTION

- Sampling
- Check sheets

DATA REPRESENTATION/ANALYSIS OF CAUSES OF PROBLEM

- Pareto charts (histograms representing subproblems in descending order of importance)
- Scatter diagrams
- Stratification
- Cause and Effect Diagrams (also known as Ishikawa or Fishbone diagrams; these diagrams represent method, manpower, material, and machine factors)

SOLUTION FORMULATION

- Brainstorming
- Consensus through discussion

TRACKING OF RESULTS

- Control Charts (graphing of data against standards and acceptable limits)

¹² J.D. Couger, "Circular Solutions," *Datamation*, January 1983, p. 136.

Other Group Decision-Making Techniques

A number of other techniques are in common usage in group decision situations of all kinds. Among the most well known are the Delphi technique and the Nominal Group Technique (NGT). A less well known but promising technique for group decision-making is Social Judgment Analysis (SJA). Brief descriptions of each of these processes is given as follows.

- **Delphi Technique** - The Delphi technique (see Linstone and Turoff, 1975, for a detailed review) was developed over a 30-year period at the Rand Corporation in an effort to improve the accuracy of forecasts of technological change. It has grown rapidly in popularity, and has been used not only for forecasting but also for policy formulation. The Delphi technique is primarily used with groups whose members are geographically remote from each other. This technique involves the following steps.
 1. The judgments of a group of individuals are captured via questionnaire.
 2. Questionnaire results are summarized by a facilitator in both statistical and thematic form.
 3. Results from the first questionnaire are sent to participants along with a new questionnaire. With the perspective of the current group judgment as background, each participant responds to the new questionnaire.
 4. Steps 2 and 3 are repeated until a group consensus emerges.
- **Nominal Group Technique** - The Nominal Group Technique was developed by Delbecq and Van de Ven in 1968 (see Delbecq, Van de Ven, and Gustafson, 1975, for a detailed review). Based on research results in the fields of social psychology, management science, and social work, NGT has gained considerable recognition and has been widely applied in education, business, and government. Although there are several variations to this technique, the basic steps in the process are the following.

1. A question is posed to the group. Each group member privately lists his or her ideas in response to the question.
 2. In round robin style, members contribute one idea each until the facilitator has listed all ideas on a flipchart.
 3. Discussion for the purpose of clarification, not evaluation, occurs. Duplicate or closely related ideas may be combined if appropriate.
 4. Members cast n votes each for the best ideas (e.g., of 20 ideas, 5 will be used, so members may vote for 5 ideas). The ideas with the most votes prevail, with number of votes determining priority or hierarchy.
- Social Judgment Analysis (SJA) - Social Judgment Analysis is the name given to field applications of Social Judgment Theory. Social Judgment Theory is associated with Hammond, Brehmer, and colleagues, and was derived from Tolman and Brunswik's approach to human cognition in terms of "probabilistic functionalism" (Tolman, 1932; Brunswik, 1943; Tolman and Brunswik, 1935; Brunswik, 1955). Social Judgment Theory holds that the integration of information to form a judgment involves *weighting* each piece of information, using *function forms* to relate pieces of information to the final judgment, and using a particular *organizing principle* for combining pieces of information to form a final judgment.

Social Judgment theorists argue that human conflict is often misunderstood as a conflict in motive or interest when it is in fact the result of differential cognitive processing of available information. They propose that disagreement can be reduced and consensus facilitated by providing people with "cognitive feedback" rather than "outcome feedback," i.e., feedback about the logic underlying judgments rather than the results of judgments alone. Laboratory research has borne out this position (Hammond, Stewart, Brehmer, and Steinmann, 1975), as have SJA field applications in such areas as labor-management negotiations, purchasing municipal land, water

resources planning, selecting police handgun ammunition, and establishing institutional objectives (Hammond, Rohrbaugh, Mumpower, and Adelman, 1977).

The fundamental requirement in SJA is that participants in group decision-making be given information about the "judgment policies" of peers (i.e., the assumptions underlying the reasoning process) before interactive decision-making begins. In other words, SJA orients an individual as to the relationship between his or her own basis for judgments and that of peers.

Comparative Research on Group Decision-Making Techniques

Research on group decision-making is primarily of two types: (1) experimental research with ad hoc heterogeneous groups working under laboratory conditions, and (2) survey research done in applied settings on ongoing homogeneous work groups functioning as quality circles or employee work teams. The groups to be used in the proposed study fit neither of the groups just described. They are, instead, ad hoc homogeneous groups, i.e., several persons from the same organization who come together to perform the tasks in this study. The findings of existing research, therefore, may be a useful background for the study, but they are not directly applicable.

A classic problem associated with group decision processes drawing heavily on individual judgment is that some individuals will invariably dominate the discussion. The dominant individuals' preferences may be accepted, with the ideas of the more introverted members of the group never voiced. Some studies suggest group discussion actually inhibits creativity (Bouchard, 1969; Holloman and Hendrick, 1972).

Another well-known problem associated with group decision making is the potential for "groupthink" (Janis, 1972). Groupthink refers to the tendency of a "mature" group (i.e., a group with stable membership and good interpersonal relationships) to suppress critical commentary in the interest of maintaining group harmony.

The Delphi technique, the Nominal Group Technique, and Social Judgment Analysis are all minimally interactive methods of group decision making that were developed to overcome the problems of individual dominance and groupthink (Steiner, 1971; Rohrbaugh, 1981). Research has shown that decision quality and participant satisfaction are higher with NGT than with the Delphi Technique (van de Ven and Delbecq, 1974; Sullivan, 1978). The Delphi technique, NGT, and SJA have been found to be equally strong with respect to decision accuracy, but SJA has been shown to be a significantly better method for reducing disagreement and developing consensus than either the Delphi technique or NGT (Rohrbaugh, 1979, 1981).

Holloman and Hendrick investigated the relative adequacy of six modes of group decision making, from the least interactive (an average of individual decisions) to the most interactive (consensus through discussion) decision-making techniques. The research was designed to determine the adequacy of the six techniques when (1) groups are homogeneous with respect to ability, (2) groups have a history of prior interaction, (3) the task is nonfactual in nature and involves a wide range of possible solutions, and (4) time limits are uniform.

The research situation involved viewing of the first 38 minutes of the film "Twelve Angry Men," which depicts the deliberations of a jury at the end of a murder trial. The initial vote of the jury, after a preliminary discussion, is 11 to 1 for guilty. The film was stopped at the point where the jury is preparing for its second vote. The subjects were told that during the remainder of the film the jurors who had initially voted guilty changed their vote, one by one, resulting in a final vote of 12 to 0 for not guilty. The research task was for each group to predict the order in which the jurors would change their votes from guilty to not guilty.

The six modes of decision-making used in the study are listed as follows in ascending order of their efficacy in performing the research task.

- Averaged Decision - Group members privately recorded their own judgments. Individual judgments were averaged to produce a group average.
- Leader Decision - Group members privately recorded their own judgments. They then selected a leader, whose judgment was used as the group judgment.
- Committee Decision - Group members privately recorded their own judgments. They then selected two leaders, who discussed their judgments with each other in private and arrived at a consensus of two.
- Majority Vote - Group members privately recorded their own judgments. The group then had a brief discussion of their judgments that concluded with a majority vote.
- Consensus - Group members privately recorded their own judgments. They then discussed their judgments and reached a consensus.
- Consensus after Majority Vote - Same process as Majority Vote, but continued discussion until a consensus was reached.

As portrayed in Figure 2, the findings showed that the decision accuracy was positively related to the quantity of interaction.

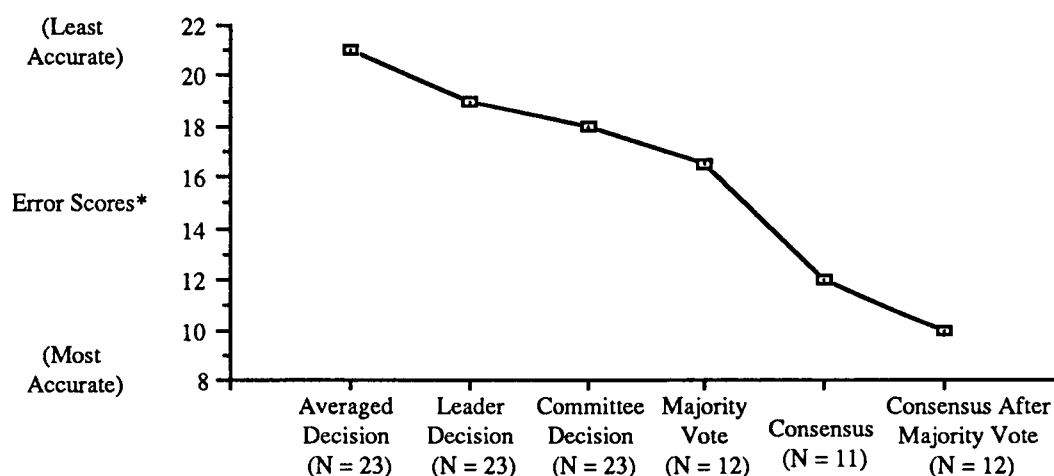


Figure 2. Accuracy of Group Decisions as a Function of Interaction Pattern¹³

* The Error Score was the sum of the absolute differences between a decision and the correct solution.

¹³ C.R. Holloman and H.W. Hendrick, "Adequacy of Group Decisions As a Function of the Decision Making Process," *Academy of Management Journal*, 15, 1972, p. 180.

Conclusions from Research on Group Decision-Making Techniques

Effective group decision-making techniques must accomodate both the advantages and disadvantages of group dynamics. On one hand, there must be sufficient interaction to facilitate the examination, clarification, and evaluation of ideas. On the other, there must be sufficient structure to ensure that the ideas of *all* group members are included for consideration.

The Social Judgment Analysis technique of apprising group members of each other's views before the group session has been shown to reduce disagreement and build consensus without loss of decision accuracy. The Consensus After Majority Vote technique in Holloman's study has also been shown to be effective in using group interaction to yield accurate decisions. Because of the apparent effectiveness of both these procedures, a melding of the two was used as a framework for group process in the current study.

"Rules" for Software Quality

Textbook Rules. There is no shortage of "rules" associated with software design, programming practices, maintenance guidelines, etc., in computer science. Nearly every introductory text on programming contains a number of "rules" that are stated, illustrated, and summarized in the form of a rule set.

One such rule set comes from *The Elements of Programming Style* by Kernighan and Plauger (1979), a standard reference text in computer science. This set of 61 mostly language-independent rules is representative of rule sets that are almost unavoidably encountered by practitioners who have had formal schooling in computer science. The authors acknowledge that their rules reflect the distilled wisdom of pioneering work done by many people. They credit D.D. McCracken and G.M. Weinberg with having long

taught the virtues of simplicity and clarity, and E.W. Dijkstra and H.D. Mills for work on structured programming that has enabled proper specification of flow of control.

The Role of Aesthetics. Several authors who have considered the value and composition of rule sets have observed a dominance of the role of aesthetics in such rule sets. Nineteen of Kernighan and Plauger's rules may be considered to be aesthetic rules.¹⁴

Leventhal (1987) addressed the role of aesthetics in her dissertation on perceptions of software quality. In expressing preferences between program segments, her "experts" focused on the abstract, i.e., problem representation, while the novices in her study focused on more concrete aspects of solution presentation.

In interviews with expert programmers whom he called "superprogrammers," Molzberger (1984) also found aesthetics to be a dominant theme related to perceptions of quality. These experts required that programs be both optically pleasing and functionally acceptable. Their aesthetic orientation was so strong, he claimed, that they visualized programs wholistically as three-dimensional structures.

Research on Rules. At a more operational level, Soloway and Ehrlich (1984) have stated "rules of programming discourse" that they believe are used by expert programmers. The context of these rules is described by them as follows.

We suggest that expert programmers have and use two types of programming knowledge: 1) *programming plans*, which are generic program fragments that represent stereotypic action sequences in programming, and 2) *rules of programming discourse*, which capture the conventions in programming and govern the composition of the plans into programs.¹⁵

¹⁴ L.M. Leventhal, Perception of Software Quality: Aesthetics and Expertise (University of Michigan, 1987), p. 15.

¹⁵ E. Soloway and K. Ehrlich, "Empirical Studies of Programming Knowledge," *IEEE Transactions on Software Engineering*, September 1984, p. 595.

Five such "rules of programming discourse" were presented in their article.

- (1) Variable names should reflect function.
- (2) Don't include code that won't be used.
- (2a) If there is a test for a condition, then the condition must have the potential of being true.
- (3) A variable that is initialized via an assignment statement should be updated via an assignment statement.
- (4) Don't do double duty with code in a non-obvious way.
- (5) An IF should be used when a statement body is guaranteed to be executed only once, and a WHILE used when a statement body may need to be repeatedly executed.¹⁶

In one study, Soloway and Ehrlich presented two versions of programs to expert and novice subjects: one "plan-like" (i.e., that followed the rules of programming discourse), and one "unplan-like" (i.e., that violated one of the rules). Each program had one blank line, and the research task was to "fill in the blank." The novices' performance reflected equal comprehension of the plan-like and unplan-like programs. The experts' performance, however, was markedly poorer with the unplan-like programs. The researchers concluded that expert programmers do, in fact, operate under a rule set, and their performance is affected if rules are violated.

In another study by the same researchers, professional programmers were shown plan-like and unplan-like programs three times and asked to recall the programs verbatim after each showing. After the first showing, 42% of the lines in the plan-like programs were recalled and 24% of the lines in the unplan-like programs were recalled. In succeeding showings, both the rate and accuracy of recall continued to be higher for the

¹⁶ Ibid, p. 597.

plan-like programs. The researchers again concluded that violation of rules had a significant detrimental effect on programmer performance.

An earlier experiment by Shneiderman (1976) also found a relationship between "proper" programs and expert recall. In his experiment, a FORTRAN program was keypunched and printed on a line printer. The card deck was shuffled, then printed again. The proper program and the shuffled program were given to a range of subjects from nonprogrammers to expert programmers. Subjects were asked to memorize, then recall the programs. The ability to memorize the proper program increased with increasing programmer expertise, while the ability to memorize the shuffled program showed minimal change across subjects. The researchers believed their results showed that expert programmers have a set of syntactic and semantic rules by which they gauge meaning and quality.

Conclusions. Empirical research seems to validate the notion that programmer expectations, or rules, do exist. Which rules, for what applications, and in what environments, however, are not at all clear.

Whatever the rules that govern programmer expectations, it appears that functional and aesthetic considerations are prominent. The relative importance of the functional and aesthetic aspects of programmer expectations, and the interaction between them, is, again, not clear.

Research on locally-derived rule sets will enable the articulation of rule sets for given environments by programmers who work in those environments. These "experts" will propose rules rather than react to rules, and will derive them from judgments about real programs rather than contrived research events.

CHAPTER III

RESEARCH PROCEDURES

The purpose of this study was to determine whether a specified group process will reliably produce a software quality "rule set" that is a valid representation of an organization's local standards of quality. The investigations concerned

1. construct validity (is the effect of *local derivation* of rules really being studied?),
2. criterion validity (can the rules be used to produce results that correlate with a known valid criterion?), and
3. reliability (are valid results consistently obtained in successive uses of the process?).

Overview of Method

A series of controlled experiments was performed in which the research task was to rank a set of modules from a given ("local") organization in order of quality. The use of a locally-derived software quality rule set in the task was the independent variable, and the resulting rank order of the modules was the dependent variable. Rankings by the experimental and control groups were compared with the ranking by the jury that derived the local rule set in order to determine the effect of the locally-derived rule set in enabling subjects to simulate the judgment of the jury.

The investigation of the validity of the rule sets was conducted in two phases: two experiments were conducted using rule sets derived in the research done by Binder, and two experiments were conducted using rule sets derived in a new way. Each experiment employed a posttest only control group design.

There was some variation in the process used to develop the four rule sets that were the objects of the investigations of validity. Separate conclusions were drawn about the reliability of the process used in Binder's exploratory work and the reliability of the more structured process used to derive new rule sets in this study. The consistency of the findings across the four tests of validity was examined as an indication of the sensitivity of the process used for deriving rule sets.

Process for Derivation of Rule Sets

The rule sets used in the study were derived in four organizations. Two organizations---the Office of Information Systems at the University of Tennessee, and Computer Technology and Imaging, Inc., in Knoxville, Tennessee---participated in Binder's research (Binder, 1987, 1988c), and two others---Science Applications International Corporation in Oak Ridge, Tennessee, and Knoxville Utilities Board in Knoxville, Tennessee---participated in the current study.

The two rule sets derived in Binder's work and the two new rule sets derived in the current study were derived through some common procedures and some differing procedures. Following is a description of how the rule sets were derived.

Common Procedures. A computer scientist acted as a planner and facilitator throughout the process. The procedures for jury selection and sampling of modules that were proposed by Poore and used by Binder were also used in the derivation of new rule sets. Briefly, these procedures were as follows.

- Jury selection - The subject organization selected four to six employees who were representative of the programming staff.
- Module selection - A random number between 1 and x was generated, where x was either the total number of lines of code or (for organizations that did not use internal subroutines) the total number of source files in the inventory of code for the subject

organization. The module associated with the random number was selected for inclusion in the sample.

- Sample size for modules - The Central Limit Theorem suggests that a sample size of 30 from a fairly normally distributed population is adequate.¹⁷ Samples of at least 36 were used.

Rule Set Derivation in Binder Experiments.¹⁸ The rule sets produced in Binder's research (1987, 1988c) and used in this study were derived by the University of Tennessee Office of Information Systems (OIS) and Computer Technology and Imaging, Inc. (CTI).

Both the OIS and CTI rule sets consist of restatements of "reasons for placing modules in their respective ranks" that were given by jury members during the group ranking of modules.¹⁹ The jury members often expressed their reasons in informal comments and incomplete sentences during the ranking process, and the reasons were rephrased by Binder as rules. Binder wrote the jury members' comments on copies of the modules being reviewed, and then later developed the restatements, i.e., rules, with primary concern for preserving the content of the jury members' comments.

In the OIS experiment, the modules were placed in the jury's rank order of quality and grouped in four equal sets: the top fourth, the second fourth, the third fourth, and the bottom fourth. Rules arising from each group were extracted, forming a rule set with four categories of rules. The top group, "Rule Group 1," was given a weight of four; "Rule Group 2," a weight of three; "Rule Group 3," a weight of two; and "Rule Group 4," a weight of one. The resulting OIS rule set is given in Appendix A.

¹⁷ L. Ott, An Introduction to Statistical Methods and Data Analysis (New York: PWS-KENT Publishing Company, 1988), p. 109.

¹⁸ L.H. Binder, personal interview, 2/13/89.

¹⁹ L.H. Binder, "Local Quality Metric for Real-Time FORTRAN Subroutines," University of Tennessee Technical Report, April 1988, p. 3.

A different method was used for developing the final form of the CTI rule set in Appendix B. Each instance of the application of a rule was given a weight based on the jury's final ranking of the module to which the rule was applied. There were 40 modules in the CTI experiment, and the final weighting of rules was reported by Binder as follows.

The cumulative weight of a rule is determined by the number of times it is used as a criterion for judging a module and by the rank of the module at which the rule is applied. For example, the rule "document the limitations of a module" was applied in modules ranked 13, 14, and 15, yielding a weight of $(41-13) + (41-14) + (41-15) = 81$. In general,

$$\text{weight} = (41-r_1) + \dots + (41-r_n)$$

where n represents the number of applications of a rule, and r_i is the rank of the module at a particular application of the rule.²⁰

Derivation of New Rule Sets. The process for deriving new rule sets in this study combined aspects of the Social Judgment Analysis and Consensus After Majority Vote techniques described in Chapter II. The new process consisted of three parts: preliminary evaluation, the group process, and final review. The three parts took place during three consecutive weeks. A pilot session with university students was conducted to rehearse the mechanics of the process before it was used in the new experiments.

Preliminary Evaluation by Individual Jurors

A group orientation meeting, an individual work session, and a group feedback meeting are involved in the preliminary evaluation. The activities in this phase are as follows.

²⁰ Ibid.

1. *Orientation meeting.* In a one-hour orientation session, a general discussion of software quality is held and the purpose and mechanics of the process are explained. The sample of modules and the Preliminary Evaluation form in Figure 3 are distributed, and jurors are given instructions on how to carry out their individual evaluations.
2. *Individual work.* Individual evaluations are conducted in the four steps that are indicated on the Preliminary Evaluation form.

Step 1: Drawing on the discussion in the orientation meeting and his or her own experience, the juror defines "software quality" in terms that are personally meaningful and relevant in the local environment.

Step 2: The juror sorts the modules and records the reasons for his or her decisions. The sorting and recording proceed as follows.

- The first module is examined and evaluated as "good" or "not so good."
- The reasons for the evaluation are written as "rules," as though the juror were instructing a junior staff member in good programming practices.

Not this...

"This module is too long"

"I don't know what 'VAR_X' is"

"It's too hard to read."

"Why is '30' used in the condition?"

But this...

"A module should not exceed X lines."

"Implement one function per module."

"Use meaningful variable names."

"Indent nested control structures X spaces."

"Skip a line between control structures."

"Avoid 'magic numbers' (numeric literals)."

"Use meaningful names for constants."

Preliminary Evaluation

<i>Step 1: Define "software quality" in your own words as it applies to your environment.</i>	
Software Quality is _____ _____	

<i>Step 3: Assign each rule a weight of 1, 2, or 3, with 1 being most important.</i>	Weight	<i>Step 2: Sort the modules into two equal groups: "good" and "not so good." Write the reasons for your decisions below. State your reasons as "rules." Repeat the process for each of those two groups. End with four equal groups.</i>

<i>Step 4: Write the numbers of the modules in the four groups. The highest quality group is level "1."</i>	
1	
2	
3	
4	

Figure 3. Preliminary Evaluation Form

- The second module is examined and evaluated as "good" or "not so good," with reasons written as rules. (Repeated uses of a rule are indicated by marking the rule. Marks are considered later in assigning weights to rules.) The evaluation continues until the modules are in two stacks, "good" and "not so good."
- Modules are upgraded or downgraded until the two stacks are of equal size. The recording of reasons as rules continues during the upgrading and downgrading.
- The process is repeated for each of the two equal stacks.
- At the end of the process, the modules will have been sorted into four equal stacks representing four different levels of quality.

Step 3: Each rule is assigned a weight of 1, 2 or 3, with 1 being most important.

Step 4: The numbers of the modules in the four stacks are recorded on the form.

The Preliminary Individual Evaluation form is returned to the facilitator at the end of the individual work period, but the juror keeps his/her copy of the modules. Jurors should be encouraged to make personal notes on the modules during the evaluation process. The notes will jog the juror's memory during the later group process and enable the group to move more rapidly.

3. *Feedback meeting.* The facilitator prepares a Composite Preliminary Evaluation so that each juror can see how his or her reasoning and judgment compares with that of the other jurors. The Composite Preliminary Evaluation has three parts.
 - Summary of Software Quality Definitions: listing of individual definitions, grouping of "factors" in individual definitions by theme, and synthesis of themes in a draft group definition.
 - Summary of Software Quality Rules: listing of individual rules grouped by weight, and listing of individual rules grouped (as much as possible) according to

themes in the software quality definition.

- **Summary of Module Evaluations:** raw data recorded on the Individual Evaluation of Modules form in Figure 4, and composite data presented on the Agreement Pattern form in Figure 5.

A sample Composite Preliminary Evaluation is given in Appendix C.

Jurors are asked to review the Composite Preliminary Evaluation before the next group meeting and identify any statements that are unclear to them. Jurors are also asked to arrange their modules in the columnar order given on the Agreement Pattern form to facilitate the group ranking session.

Group Process

The group decision-making process begins with a review of individual definitions and rules about software quality. The ranking of modules occurs next, and is facilitated by the agreement pattern that emerged from individual evaluations. Finally, a draft rule set is developed from the summary of rules.

1. *Review of Individual Definitions and Rules.* The facilitator reads the draft definition of software quality synthesized from individual definitions and asks for comments or questions. The purpose of discussion at this stage is clarification, not agreement. In similar fashion, the facilitator asks for comments or questions about the rules. The facilitator acknowledges that the rules are in unedited form, and reminds the group that the drafting of the rule set will take place in a separate meeting. Questions and observations are entertained, and any difference in views is acknowledged and tabled for later discussion.

Individual Module Ranks

Modules	Jury Members				Agreement Code *
	1	2	3	4	
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					
19					
20					
21					
22					
23					
24					
25					
26					
27					
28					
29					
30					
31					
32					
33					
34					
35					
36					

* Agreement Codes:	
A	Agreement
NA ₁	Near Agreement (1 juror differs)
NA ₂	Near Agreement (2/2 split - adjacent cats.)
D	Disagreement
#	Highest Mode

Figure 4. Individual Module Ranks Form

Agreement Pattern

Quality Level	Agreements all jurors agree	Near Agreements 1 or 2 jurors differ	Disagreements 3 or more jurors disagree
1			
2			
3			
4			

Figure 5. Agreement Pattern Form

2. *Ranking Modules.* Modules are ranked by subgroups in the cells of the Agreement Pattern form, then integrated to form the final ranking. Statements of rationale ("rules") are given for each ranking decision, with the facilitator recording each statement for later integration into the rule set. Ranking within each subgroup proceeds as follows, with all decisions by consensus.

- Ranking begins by comparing any two modules. The better module is module "A" and the other is module "B."
- A third module, "X," is then compared with A.
- If X is better than A, then the new rank order is X, A, B.
If X is not better than A, then X is compared with B.
If X is better than B, then the new rank order is A, X, B.
If X is not better than B, then the new rank order is A, B, X.
- Each remaining module in the subgroup is compared with the ranked modules in this top-down manner until all modules in the subgroup have been ranked.

Subgroups are ranked in the following order of cells in the Agreement Pattern:

- Agreements, Quality Level 1
- Agreements, Quality Level 2
- Agreements, Quality Level 3
- Agreements, Quality Level 4
- Near Agreements, Quality Level 1
- Near Agreements, Quality Level 2
- Near Agreements, Quality Level 3
- Near Agreements, Quality Level 4
- Disagreements

The "boundaries" between subgroups within a category are checked during the process to ensure proper order. After the subgroup "Agreements, Quality Level 2" (A-2) has been ranked, for example, its highest member will be compared with the lowest member of "Agreements, Quality Level 1" (A-1) to verify that the lowest member of A-1 is, in fact, better than the highest member of A-2. If not, comparison continues until the proper order is established.

The final module order is obtained by integrating the newly-ranked sets of Agreements (set A, modules $A_1...A_x$), Near Agreements (set NA, modules $NA_1...NA_y$), and Disagreements (set D, modules $D_1...D_z$). Set $A_1...A_x$ is the reference set, and other modules are inserted as appropriate. The integration of the sets proceeds as follows.

- Module NA_1 is compared with the each module in set A (starting with A_1) until NA_1 's proper place in the the rank order of set A is found.
 - Module NA_2 is compared with each module in set A (starting with the module below NA_1) until NA_2 's proper place in the the rank order of set A is found.
 - The procedure continues until either a module in set NA has been placed after module A_x (i.e., the end of set A is reached) or module NA_y has been placed in set A (i.e., the end of set NA is reached).
 - Sets A and NA now form a newly-integrated single set of modules in rank order of quality. Set D is integrated into the new set using the same procedure just used to integrate NA into A.
3. *Drafting the Rule Set.* After the group ranking meeting, the facilitator prepares a draft rule set for the group to consider at the group's rules meeting. The draft rule set

(1) incorporates comments from the previous meeting about the software quality definition and raw listing of rules, (2) includes new rules arising from the ranking process, and (3) eliminates redundancy. The draft rule set is a point of departure for discussion. The group discusses the draft until a preliminary consensus emerges.

Final Review

In the final review session, the revised definition of software quality and the rule set are discussed until final forms are established by consensus.

Schedule of Events

Following advance activity relative to jury selection, sampling of modules, and scheduling, the group process is conducted as follows.

Week 1

Orientation meeting (1 hr.)	Overview of process. Distribution of modules and Preliminary Evaluation form.
Individual work (3 hrs.)	Same day as orientation. Work returned at day's end.
Staff work (1 day)	Preparation of Composite Preliminary Evaluation.
Feedback meeting (1 hr.)	Composite Preliminary Evaluation distributed.

Week 2

Ranking meeting (1 day)	Software quality definitions and rules reviewed. Modules ranked.
Staff work (1 day)	Preparation of draft rule set.
Rules meeting (3 hrs.)	Review of draft rule set.
Staff work (4 hrs.)	Revision of draft rule set.

Week 3

Rules meeting (2 hrs.)	Final version of rule set developed through consensus.
Staff Work (2 hrs.)	Preparation of final rule set.

Experimental Design for Testing of Rule Sets

The validity of each rule set was investigated using a "true experimental design."²¹ The design was a "posttest only control group design," which is usually used in situations that cannot be pretested. The independent variable in the design was the use of a locally-derived rule set, and the dependent variable was the order in which software modules were ranked by the subjects.

Four subject groups (one experimental and three control) used different approaches to rank the modules: one used locally-derived rules (rules derived in the organization whose modules were being evaluated), one used "generic" rules (Kernighan and Plauger's rules in *The Elements of Programming Style*, given in Appendix D), one used "placebo" rules (either the nonhelpful rules in Appendix E or rules from another organization), and one used personal judgment alone.

Each of the subjects reviewed the modules and placed them in rank order of quality. The ranks assigned by subjects in each group were averaged to produce a group ranking. The rank order by each group was compared with the rank order by the jury that had originally reviewed the modules and derived the local rule set.

Construct validity was determined by examining the relative similarity of the module ordering by each of the test groups with the module ordering by the jury. Following is the

²¹ P.D. Leedy, Practical Research: Planning and Design (New York: MacMillan Publishing Company, 1985), p. 213.

rationale used for drawing inferences about construct validity.

- If, in each experiment, the group using local rules produces a module order with the greatest similarity (i.e., the highest correlation) to the jury's ordering, then the construct under examination is indeed likely to be "locally-derived rules."
- If the groups using local and generic rules produce rankings similar to the jury's and the placebo and no rules groups do not, the construct may simply be "rules" rather than "local derivation of rules."
- If all groups using a rule set produce rankings similar to the jury's and only the no rules group does not, the construct may simply be "perception of help."
- If all groups produce rankings similar to the jury's, no conclusions may be drawn.

The jury's ordering of modules is the known valid criterion against which test groups' ranks will be compared. If the correlation between the judgments of the group using locally-derived rules and the jury is both higher than the correlation between other groups and the jury and statistically significant as well, the rule set is likely to have represented the organization's sense of quality in a valid way.

The experimental design is represented in Figure 6.

Subjects

Subjects who comprised the experimental and control groups were not associated with the organizations in which rule sets used in the research were derived.

Subjects were either upperclass undergraduate students in computer science, business, or engineering; graduate students in computer science; or practitioners with at least two years of experience. This population might well describe the pool of candidates for employment in the software organizations involved in the study. Since new employees would surely be among the people to whom an organization would like to be able to convey its sense of quality, this was an appropriate subject population.

<i>Event</i>		<i>Outcomes</i>
Jury of Experts reviews modules	----->	Ranking of Modules (O_J , criterion for all tests)
	----->	Rule Set
Experimental Group of Naive Programmers reviews modules, using Local Rule Set	----->	Ranking of Modules (O_L)
Control Group of Naive Programmers reviews modules, using Generic Rule Set	----->	Ranking of Modules (O_G)
Control Group of Naive Programmers reviews modules, using Placebo Rule Set	----->	Ranking of Modules (O_P)
Control Group of Naive Programmers reviews modules, using No Rule Set	----->	Ranking of Modules (O_C)
<i>Analysis</i>		
O_L and O_J are compared to yield Correlation_{LJ} .		
O_G and O_J are compared to yield Correlation_{GJ} .		
O_P and O_J are compared to yield Correlation_{PJ} .		
O_C and O_J are compared to yield Correlation_{CJ} .		
Correlations $_{LJ}$, $_{GJ}$, $_{PJ}$, and $_{CJ}$ are compared to yield the experimental effect.		

Figure 6. Posttest Only Control Group Design

Data Analysis

The Spearman rho rank-difference correlation coefficient was computed to assess the relationship between module orders produced by individuals within each group, and between each group's average ranks and the jury's rank ordering. A normal Z deviate was used for a test of significance. Individual/individual correlations enabled examination of within-group interrater reliability. A confidence interval around the group/jury correlations enabled examination of between-group differences.

CHAPTER IV

RESULTS

Tests of construct and criterion validity were performed for four different rule sets. The rule sets were derived in (1) the Office of Information Systems at the University of Tennessee (OIS), (2) Computer Technology and Imaging, Inc., (CTI) in Knoxville, Tennessee, (3) Scientific Applications International Corporation (SAIC) in Oak Ridge, Tennessee, and (4) Knoxville Utilities Board (KUB) in Knoxville, Tennessee.

Each test of a rule set involved four groups of subjects that used either locally-derived rules, generic rules, placebo rules, or no rules to rank a sample of modules in order of quality. The sample of modules was the same sample that had been ranked by the jury that developed the rule set being tested.

Three data sets are presented for each test.

1. Raw Data - ranks by individual subjects and the average ranks for each group.
2. Individual Correlations - The Spearman rho (r_s) rank-difference correlation coefficient was computed for each pair of subjects within a group, and for each subject and the jury. A normal Z deviate was used for a test of significance.
3. Group Correlations - The Spearman rho (r_s) rank-difference correlation coefficient was computed for each group and the jury. A normal Z deviate was used for a test of significance.

Conclusions about reliability and validity are drawn from patterns in the data across tests. Interrater reliability is examined through individual correlations, and construct and criterion validity are examined through group correlations.

Test of Validity for OIS Rule Set

In the test of the OIS rule set, 36 modules from the OIS inventory of 516 batch COBOL programs were ranked by 20 subjects. Each of four test groups had five subjects who were upperclass business students at the University of Tennessee. All subjects had taken at least six hours of academic coursework in computer science and had just completed a course in the COBOL programming language.

The CTI rule set was used by the group using "placebo" rules.

The ranks assigned by individual subjects and the average ranks for groups are given in Table 2. The correlations between the ranks assigned by individuals within a group and between individuals and the jury are given in Tables 3 through 6. The correlations between average ranks assigned by groups and ranks assigned by the jury are given in Table 7.

Findings. While there was a highly significant correlation (at least $p < .001$) between the module order produced by all groups and the rank order by the OIS jury, the correlation between the group using the locally-derived rule set and the jury was highest.

Construct validity is examined by comparing the relative strength of correlations between each group and the jury. Although all correlations were high, the group using local rules was best able to simulate the judgment of the jury. The modules used in this test varied widely in quality, and high correlations between reasonably well-qualified evaluators are not surprising when differences in modules are sharp. Even so, the stronger correlation between the group using the OIS rule set and the OIS jury indicates that the use of locally-derived rules indeed had a differential effect in aligning the thinking of the naive programmers with that of the jury. Construct validity, therefore, was generally supported in this test.

Criterion validity is examined by comparing the module order by the group using the locally-derived rule set with the module order by the jury. The correlation of .775 between the two orders provides strong evidence of criterion validity in this test.

Table 2. Rank Ordering of OIS Modules: Raw Data

Jury Order	Local Rules						Generic Rules						Placebo Rules						No Rules									
	1	2	3	4	5	Avg. Rank	1	2	3	4	5	Avg. Rank	1	2	3	4	5	Avg. Rank	1	2	3	4	5	Avg. Rank				
1	3	9	12	4	8	7.2	2	17	6	9	15	20	13.4	10	2	4	1	22	2	6.2	2	1	22	8	8	1	8.0	4
2	9	10	18	14	12	12.6	11	5	7	7	24	3	9.2	4	3	16	17	36	31	20.6	25	32	20	17	16	9	18.8	22
3	8	6	8	24	7	10.6	7	7	11	5	26	5	10.8	6	7	13	21	20	10	14.2	10	10	8	16	13	19	13.2	10
4	5	8	9	11	4	7.4	3	16	3	3	19	1	8.4	2	4	5	3	28	5	9.0	3	4	19	14	4	6	9.4	5
5	7	2	14	3	2	5.6	1	6	5	8	17	2	7.6	1	1	3	6	19	1	5.8	1	3	2	6	1	8	4.0	2
6	6	20	31	7	9	14.6	13	3	9	4	1	25	8.4	2	6	9	15	23	29	16.4	16	2	4	7	2	2	3.4	1
7	13	14	1	1	19	9.6	5	11	2	14	21	28	15.2	13	22	6	29	3	14	14.8	12	21	5	12	29	5	14.4	13
8	4	22	30	8	11	15.0	14	4	8	21	25	15	14.6	12	8	7	20	21	35	18.2	21	14	9	15	10	18	13.2	10
9	20	18	5	17	26	17.2	17	28	18	28	30	18	24.4	28	25	20	2	11	8	13.2	7	17	24	24	17	22	20.8	25
10	1	5	4	25	5	8.0	4	8	4	10	23	13	11.6	7	9	8	19	2	33	14.2	10	27	3	13	15	12	14.0	12
11	21	7	7	13	3	10.2	6	22	28	19	20	12	20.2	24	20	21	10	10	17	15.6	14	5	13	26	18	4	13.2	10
12	26	4	3	15	6	10.8	8	29	22	26	31	33	28.2	34	24	22	13	5	4	13.6	8	12	15	23	19	20	17.8	20
13	27	3	13	28	27	19.6	22	33	30	31	29	23	29.2	35	23	31	11	33	19	23.4	28	19	16	22	14	31	20.4	24
14	10	1	32	12	23	15.6	16	14	16	1	28	21	16	15	18	1	22	2	16	11.8	6	11	21	1	6	25	12.8	8
15	29	11	2	23	1	13.2	12	2	14	2	18	16	10.4	5	11	11	4	12	11	9.8	4	24	25	4	3	24	16.0	16
16	22	19	24	29	22	23.2	26	35	24	30	32	17	27.6	32	10	18	18	32	20	19.6	22	18	17	25	30	32	24.4	27
17	23	17	25	18	25	21.6	25	26	27	23	5	7	17.6	17	13	19	9	1	9	10.2	5	15	23	18	12	11	15.8	15
18	15	12	15	32	16	18.0	18	15	1	12	16	34	15.6	14	15	14	31	15	13	17.6	19	13	7	5	24	10	11.8	7
19	14	13	6	6	15	10.8	8	34	10	20	9	27	20.0	23	14	12	28	7	24	17.0	18	22	12	10	22	14	16.0	16
20	11	28	22	9	21	18.2	20	10	15	13	12	11	12.2	8	5	2	30	18	15	14.0	9	7	1	3	11	3	5.0	3
21	25	16	17	19	29	21.2	23	30	36	27	33	8	26.8	30	31	28	8	17	26	22.0	26	20	36	33	25	16	26.0	28
22	24	24	26	21	28	24.6	27	23	33	22	14	6	19.6	20	33	23	7	13	3	15.8	15	16	18	19	20	15	17.6	18
23	36	33	27	34	31	32.2	35	24	21	36	4	35	24.0	27	19	33	35	6	32	25.0	30	25	14	36	32	30	27.4	29
24	17	23	35	35	17	25.4	28	9	13	11	27	29	17.8	18	16	15	16	8	34	17.8	20	9	11	2	5	23	10.0	6
25	18	31	10	2	32	18.6	21	19	19	18	2	24	16.4	16	17	35	32	21	12	23.4	28	28	6	20	21	13	17.6	18
26	19	27	11	20	13	18.0	18	12	23	24	6	4	13.8	11	26	27	14	9	7	16.6	17	23	30	29	9	7	19.6	23
27	31	34	33	22	33	30.6	34	27	12	17	36	36	25.6	29	36	25	26	34	27	29.6	34	30	35	28	36	36	33.0	36
28	33	26	28	30	24	28.2	30	13	35	25	7	19	19.8	22	28	34	12	4	23	20.2	24	29	27	21	27	34	27.6	30
29	2	21	16	10	10	11.8	10	1	26	6	22	9	12.8	9	21	10	5	35	6	15.4	13	6	29	9	7	26	15.4	14
30	12	25	34	16	20	21.4	24	20	20	15	13	30	19.6	20	12	17	24	25	21	19.8	23	8	10	35	23	17	18.6	21
31	34	32	21	31	30	29.6	31	32	17	34	35	22	28.0	33	27	24	27	29	18	25.0	30	33	26	30	33	29	30.2	31
32	16	15	23	5	18	15.4	15	18	34	16	3	31	20.4	25	29	29	36	14	28	27.2	32	26	28	11	28	21	22.8	26
33	35	36	19	26	35	30.2	32	25	29	33	10	14	22.2	26	32	30	33	16	25	27.2	32	35	31	27	34	28	31.0	32
34	32	29	20	36	34	30.2	32	36	32	29	11	26	26.8	30	35	36	23	31	36	30.2	36	36	33	34	26	27	31.2	33
35	30	30	29	33	14	27.2	29	31	31	32	34	32	32.0	36	30	26	34	27	22	27.8	33	34	34	32	31	33	32.8	34
36	28	35	36	27	36	32.4	36	21	25	35	8	10	19.8	22	34	32	25	30	30	30.2	36	31	32	31	35	35	32.8	34

Table 3. OIS Rule Set Test: Individual Correlations for Local Rules Group

	Subject 1	Subject 2	Subject 3	Subject 4	Subject 5
Subject 1					
r_s	-	-	-	-	-
Z					
Subject 2					
r_s	.533	-	-	-	-
Z	3.151				
Subject 3					
r_s	.199	.534	-	-	-
Z	1.176	3.158			
Subject 4					
r_s	.643	.356	.303	-	-
Z	3.804	2.105	1.791		
Subject 5					
r_s	.635	.673	.456	.377	-
Z	3.757	3.979	2.695	2.228	
Jury					
r_s	.611	.758	.478	.459	.623
Z	3.617	4.486	2.829	2.718	3.685
significance of Subject/Jury corr.	$p < .0005$	$p < .0001$	$p < .005$	$p < .005$	$p < .0005$

Table 4. OIS Rule Set Test: Individual Correlations for Generic Rules Group

	Subject 1	Subject 2	Subject 3	Subject 4	Subject 5
Subject 1					
r_s	-	-	-	-	-
Z					
Subject 2					
r_s	.502	-	-	-	-
Z	2.973				
Subject 3					
r_s	.755	.615	-	-	-
Z	4.468	3.636			
Subject 4					
r_s	.203	-.100	.036	-	-
Z	1.200	-.591	.212		
Subject 5					
r_s	.342	.007	.221	.047	-
Z	2.025	.043	1.308	.277	
Jury					
r_s	.390	.644	.547	-.189	.324
Z	2.306	3.812	3.239	-1.116	1.917
significance of Subject/Jury corr.	$p < .05$	$p < .0001$	$p < .001$		$p < .05$

Table 5. OIS Rule Set Test: Individual Correlations for Placebo Rules Group

	Subject 1	Subject 2	Subject 3	Subject 4	Subject 5
Subject 1					
r_s	-	-	-	-	-
Z					
Subject 2					
r_s	.750	-	-	-	-
Z	4.434				
Subject 3					
r_s	.255	.288	-	-	-
Z	1.511	1.702			
Subject 4					
r_s	-.007	.117	.001	-	-
Z	-.041	.694	.007		
Subject 5					
r_s	.207	.315	.501	.119	-
Z	1.227	1.865	2.966	.702	
Jury					
r_s	.733	.663	.493	.129	.325
Z	4.337	3.920	2.918	.761	1.922
significance of Subject/Jury corr.	$p < .0001$	$p < .0001$	$p < .005$		$p < .05$

Table 6. OIS Rule Set Test: Individual Correlations for No Rules Group

	Subject 1	Subject 2	Subject 3	Subject 4	Subject 5
Subject 1					
r_s	-	-	-	-	-
Z					
Subject 2					
r_s	.523	-	-	-	-
Z	3.096				
Subject 3					
r_s	.526	.495	-	-	-
Z	3.111	2.928			
Subject 4					
r_s	.706	.395	.679	-	-
Z	4.174	2.337	4.017		
Subject 5					
r_s	.573	.562	.447	.545	-
Z	3.391	3.327	2.645	3.227	
Jury					
r_s	.574	.557	.483	.602	.608
Z	3.397	3.295	2.857	3.562	3.595
significance of Subject/Jury corr.	$p < .0005$	$p < .0005$	$p < .001$	$p < .0005$	$p < .0005$

Table 7. OIS Rule Set Test: Correlations Between Test Groups and Jury

	Locally-Derived Rules	Generic Rules	Placebo Rules	No Rules
r_s	.775	.543	.713	.683
Z	4.585	3.210	4.221	4.040
significance	$p < .0001$	$p < .001$	$p < .0001$	$p < .0001$

Test of Validity for CTI Rule Set

In the test of the CTI rule set, 40 modules from the CTI inventory of 239 real-time FORTRAN programs were ranked by 12 subjects. Each of four test groups had three subjects: one upperclass undergraduate student, one graduate student, and one practitioner. Students were enrolled at the University of Tennessee and practitioners were employees of the University.

The OIS rule set was used by the group using "placebo" rules.

The ranks assigned by individual subjects and the average ranks for groups are given in Table 8. The correlations between the ranks assigned by individuals within a group and between individuals and the jury are given in Tables 9 through 12. The correlations between average ranks assigned by groups and ranks assigned by the jury are in Table 13.

Findings. Two of the three experimental groups, the local rules group and the generic rules group, produced a module order that correlated significantly ($p < .05$) with the module order by the CTI jury. Of these two, the correlation between the group using the locally-derived rule set and the jury was highest. Both construct and criterion validity were supported in this test, therefore, due to the fact that the correlation between the group using local rules and the CTI jury was both higher than any other group and statistically significant.

Table 8. Rank Ordering of CTI Modules: Raw Data

Jury Order	Local Rules				Generic Rules				Placebo Rules				No Rules							
	U	G	P	Avg. Rank	U	G	P	Avg. Rank	U	G	P	Avg. Rank	U	G	P	Avg. Rank				
1	2	6	6	4.7	3	1	2	8	3.7	1	11	3	2	5.3	2	1	1	1.0		
2	5	3	7	5.0	4	13	4	5	7.3	5	2	15	12	9.7	7	4	5	4.7		
3	4	2	2	2.7	2	5	6	2	4.3	3	1	14	33	16.0	12	17	4	3	8.0	
4	19	11	17	15.7	18	38	27	40	35.0	38	34	36	34	34.7	37	31	19	37	29.0	
5	13	15	5	11.0	9	9	9	22	13.3	11	14	12	6	10.7	8	5	27	2	11.3	
6	28	24	13	21.7	24	10	3	17	10.0	6	20	21	31	24.0	26	15	35	9	19.7	
7	12	12	15	13.0	11	29	19	39	29.0	33	8	23	39	23.3	24	34	6	7	15.7	
8	6	1	8	5.0	4	6	13	1	6.7	4	6	13	18	12.3	9	3	23	6	10.7	
9	14	7	16	12.3	10	40	31	15	28.7	32	10	33	11	18.0	15	36	7	35	26.0	
10	32	29	28	29.7	19	16	18	29	21.0	19	23	10	17	16.7	13	27	31	26	28.0	
11	36	36	36	36.0	38	18	16	31	21.7	21	13	29	16	19.3	18	30	39	27	32.0	
12	22	35	25	27.3	25	31	22	19	24.0	27	37	7	24	22.7	23	24	40	40	34.7	
13	7	16	22	15.0	15	30	14	7	17.0	14	7	24	13	14.7	11	12	22	8	14.0	
14	1	4	1	2.0	1	8	1	3	4.0	2	3	11	10	8.0	5	2	8	15	8.3	
15	31	34	33	32.7	33	22	17	28	22.3	22	36	26	32	31.3	36	18	28	34	26.7	
16	11	5	14	10.0	7	37	23	10	23.3	26	15	32	29	25.3	30	35	2	14	17.0	
17	39	38	31	36.0	38	36	30	37	34.3	37	40	30	37	35.7	38	39	38	38	38.3	
18	30	31	32	31.0	30	17	35	9	20.3	18	22	25	23	23.3	24	23	36	25	28.0	
19	35	39	29	34.3	34	27	40	38	35.0	38	39	40	38	39.0	40	37	26	36	33.0	
20	10	18	24	17.3	20	12	26	21	19.7	17	25	22	26	24.3	28	7	17	20	14.7	
21	18	9	4	10.3	8	39	15	14	22.7	23	12	19	20	17.0	14	32	20	11	21.0	
22	29	28	26	27.7	26	19	10	18	15.7	13	28	27	8	21.0	21	20	12	29	20.3	
23	38	32	35	35.0	36	25	12	27	21.3	20	35	31	15	27.0	32	21	29	30	26.7	
24	37	37	38	37.3	40	28	11	30	23.0	24	30	28	36	31.3	36	25	25	33	27.7	
25	16	21	9	15.5	16	3	21	12	12.0	10	17	1	3	7.0	3	13	9	5	17	13.0
26	15	22	9	15.5	16	2	20	11	11.0	8	18	1	3	7.3	4	14	9	5	18	14.0
27	26	27	34	29.0	28	14	34	32	26.7	29	27	8	28	21.0	21	29	18	23	23.3	
28	20	13	20	17.7	21	35	39	36	36.7	40	32	17	27	25.3	30	38	13	22	24.3	
29	25	8	11	14.7	14	4	25	4	11.0	8	31	16	25	24.0	26	6	3	4	4.3	
30	24	19	18	20.3	23	7	7	16	10.0	6	19	34	5	19.3	18	11	16	16	14.3	
31	3	14	23	13.3	12	26	5	26	19.0	15	4	4	1	3.0	1	10	14	13	12.3	
32	33	26	37	32.0	32	15	37	24	25.3	28	26	9	7	14.0	10	19	32	28	26.3	
33	21	10	21	17.3	20	34	36	20	30.0	34	33	18	35	28.7	34	33	21	10	21.3	
34	23	40	30	31.0	30	33	32	35	33.3	36	38	39	40	39.0	40	40	30	31	33.7	
35	8	17	3	9.3	6	20	8	13	13.7	12	5	6	14	8.3	6	8	15	12	11.7	
36	9	20	12	13.7	13	11	33	25	23.0	24	21	5	30	18.7	16	9	11	19	13.0	
37	27	30	27	28.0	27	23	38	23	28.0	31	24	37	22	27.7	33	26	24	24	24.7	
38	40	25	40	35.0	36	32	24	34	30.0	34	16	20	21	19.0	17	22	37	39	32.7	
39	34	33	39	35.3	37	21	29	33	27.7	30	29	38	9	25.3	30	28	34	32	31.3	
40	17	23	19	19.7	22	24	28	6	19.3	16	9	35	19	21.0	21	16	33	21	23.3	

Table 9. CTI Rule Set Test: Individual Correlations for Local Rules Group

	Undergraduate	Graduate	Practitioner
Undergraduate			
r_s	-	-	-
Z			
Graduate			
r_s	.795	-	-
Z	4.964		
Practitioner			
r_s	.812	.791	-
Z	5.068	4.938	
Jury			
r_s	.301	.347	.370
Z	1.878	2.168	2.309
significance of Subject/Jury corr.	$p < .05$	$p < .05$	$p < .05$

Table 10. CTI Rule Set Test: Individual Correlations for Generic Rules Group

	Undergraduate	Graduate	Practitioner
Undergraduate			
r_s	-	-	-
Z			
Graduate			
r_s	.402	-	-
Z	2.513		
Practitioner			
r_s	.497	.439	-
Z	3.104	2.744	
Jury			
r_s	.130	.465	.195
Z	.811	2.907	1.220
significance of Subject/Jury corr.		$p < .005$	

Table 11. CTI Rule Set Test: Individual Correlations for Placebo Rules Group

	Undergraduate	Graduate	Practitioner
Undergraduate			
r_s	-	-	-
Z			
Graduate			
r_s	.368	-	-
Z	2.301		
Practitioner			
r_s	.471	.395	-
Z	2.942	2.466	
Jury			
r_s	.265	.103	-.040
Z	1.652	.646	-.248
significance of Subject/Jury corr.	$p < .05$		

Table 12. CTI Rule Set Test: Individual Correlations for No Rules Group

	Undergraduate	Graduate	Practitioner
Undergraduate			
r_s	-	-	-
Z			
Graduate			
r_s	.305	-	-
Z	1.902		
Practitioner			
r_s	.577	.572	-
Z	3.603	3.574	
Jury			
r_s	.119	.188	.267
Z	.742	1.177	1.665
significance of Subject/Jury corr.	$p < .05$		

Table 13. CTI Rule Set Test: Correlations Between Test Groups and Jury

	Locally-Derived Rules	Generic Rules	Placebo Rules	No Rules
r_s	.369	.312	.175	.203
Z	2.302	1.946	1.093	1.265
significance	$p < .05$	$p < .05$		

Reliability of Process Used by Binder

Both construct and criterion validity were supported in the tests of the rule sets developed in Binder's earlier work. The similarities in the test results (i.e., the strongest correlation between the local rules group and the jury, and the statistical significance of the correlation between the local rules group and the jury) suggest that the process used was reliable in producing valid results. The differences in the test results, however, raise some questions.

The correlation between the local rules group and the jury was .775 ($p < .0001$) in the OIS test, and was .369 ($p < .05$) in the CTI test. While both are statistically significant, there is considerable disparity in the strength of the correlations. The correlation represents the degree to which naive programmers were able to align their thinking with that of local experts in making judgments about software quality. The practical significance of the strong correlation in the OIS test is readily apparent; the practical significance of the correlation in the CTI test is unclear.

The pattern of correlations between the test groups and the jury also differed in the two tests. Although local rules proved to be most helpful in aligning the thinking of naive programmers with that of local experts, the relative helpfulness of generic rules, placebo rules, and no rules differed. In the OIS test, the relative helpfulness of the rules appeared to be as follows (in descending order): local rules, placebo rules, no rules, generic rules.

In the CTI test, however, the relative helpfulness of the rules appeared as follows: local rules, generic rules, no rules, placebo rules. The effect of nonlocal rules may vary across environments for the same reason that the use of global software quality metrics is unpredictable, i.e., that local variables related to application, language, compiler, culture, etc., are essential factors in calibrating an evaluation of software quality in a given organization.

Test of Validity for SAIC Rule Set

The SAIC organization providing data processing support for the U.S. Department of Energy's Office of Scientific and Technical Information (OSTI) was the field site in which the SAIC rule set was developed. The SAIC/OSTI rule set is given in Appendix F.

In the test of the SAIC rule set, 34 modules from the SAIC/OSTI inventory of 863 FORTRAN programs were ranked by 12 subjects. Each of four test groups had three subjects: one computer science graduate student, and two industrial practitioners with at least two years of experience.

The "placebo" rules used in this test are given in Appendix E.

The ranks assigned by individual subjects and the average ranks for groups are given in Table 14. The correlations between the ranks assigned by individuals within a group and between individuals and the jury are given in Tables 15 through 18. The correlations between average ranks assigned by groups and ranks assigned by the jury are in Table 19.

Findings. All group correlations with the jury were statistically significant (at least $p < .0005$). The no rules group correlated most highly with jury, followed by the generic, local, and placebo groups. The 95% confidence intervals for the top three groups were overlapping, however, so the similarity of their judgments with the jury cannot really be distinguished. Criterion validity was supported for local rules in this test, but construct validity was not.

Table 14. Rank Ordering of SAIC Modules: Raw Data

Jury Order	Local Rules				Generic Rules				Placebo Rules				No Rules			
	G	P	P	Avg. Rank	G	P	P	Avg. Rank	G	P	P	Avg. Rank	G	P	P	Avg. Rank
1	2	8	20	11.7	27	17	4	16.0	19	13	21	17.7	8	10	14	10.7
2	5	4	14	10.3	9	6	25	13.3	20	2	24	15.3	1	9	15	8.3
3	3	11	30	14.3	29	2	20	17.0	3	29	16	16.0	15	21	1	12.3
4	14	2	6	5.7	2	8	1	3.7	18	3	15	12.0	14	1	2	5.7
5	13	5	2	7.3	12	7	3	7.3	7	9	18	11.3	21	3	3	9.0
6	22	6	21	15.3	8	4	11	7.7	21	5	4	10.0	5	6	19	10.0
7	24	3	22	14.3	3	16	10	9.7	28	6	22	18.7	7	4	11	7.3
8	8	19	32	19.7	18	20	15	17.7	14	33	8	18.3	16	19	7	14.0
9	11	23	10	19.3	13	1	19	11.0	6	14	7	9.0	12	12	8	10.7
10	18	17	28	20.3	7	24	16	15.7	12	32	10	18.0	25	27	16	22.7
11	19	9	27	18.3	22	5	2	9.7	30	12	14	18.7	3	22	28	17.7
12	7	1	24	8.3	1	3	9	4.3	17	7	5	9.7	9	5	12	8.7
13	9	22	7	11.0	23	9	6	12.7	5	1	1	2.3	19	13	6	12.7
14	6	24	4	14.0	20	12	27	19.7	1	23	9	11.0	10	17	13	13.3
15	1	15	3	9.7	5	23	13	13.7	2	22	2	8.7	4	7	27	12.7
16	16	21	15	19.0	4	15	7	8.7	9	28	29	22.0	2	11	24	12.3
17	15	13	1	11.7	25	11	18	18.0	22	4	23	16.3	28	25	17	23.3
18	26	14	25	18.3	31	28	23	27.3	15	15	19	16.3	33	23	31	29.0
19	12	7	5	8.0	24	26	14	21.3	16	11	3	10.0	6	20	5	10.3
20	20	12	9	12.7	17	22	8	15.7	8	21	13	14.0	11	8	9	9.3
21	4	20	23	14.3	16	27	17	20.0	13	26	30	23.0	17	18	23	19.3
22	30	31	31	30.3	19	29	32	26.7	31	20	33	28.0	3	1	29	27.0
23	17	29	19	23.0	6	25	28	19.7	4	27	17	16.0	24	28	20	24.0
24	34	16	11	19.7	30	30	22	27.3	26	8	6	13.3	23	26	30	26.3
25	21	27	18	23.0	14	13	30	19.0	10	30	25	21.7	22	14	18	18.0
26	31	25	29	28.0	11	21	31	21.0	24	31	26	27.0	20	24	10	18.0
27	23	10	33	22.0	15	14	21	16.7	25	16	32	24.3	13	2	4	6.3
28	10	18	8	14.0	10	10	12	10.7	23	24	31	26.0	32	15	25	24.0
29	25	26	12	18.0	28	18	5	17.0	29	10	34	24.3	29	30	26	28.3
30	33	30	34	32.3	21	34	26	27.0	11	25	11	15.7	18	16	22	18.7
31	28	28	16	22.0	33	32	24	29.7	27	17	27	23.7	30	32	29	30.3
32	32	33	13	24.3	26	19	33	26.0	34	18	28	26.7	26	33	33	30.7
33	27	34	17	24.0	32	31	29	30.7	33	34	12	26.3	34	34	34	34.0
34	29	32	26	30.7	34	33	34	33.7	32	19	20	23.7	27	31	32	30.0
				31				34				26				31

Table 15. SAIC Rule Set Test: Individual Correlations for Local Rules Group

	Graduate	Practitioner 1	Practitioner 2
Graduate			
r_s	-	-	-
Z			
Practitioner 1			
r_s	.472	-	-
Z	2.709		
Practitioner 2			
r_s	.064	.173	-
Z	.366	.996	
Jury			
r_s	.659	.744	.102
Z	3.784	4.275	.587
significance of Subject/Jury corr.	$p < .0001$	$p < .0001$	

Table 16. SAIC Rule Set Test: Individual Correlations for Generic Rules Group

	Graduate	Practitioner 1	Practitioner 2
Graduate			
r_s	-	-	-
Z			
Practitioner 1			
r_s	.426	-	-
Z	2.446		
Practitioner 2			
r_s	.327	.481	-
Z	1.876	2.764	
Jury			
r_s	.422	.653	.573
Z	2.423	3.750	3.294
significance of Subject/Jury corr.	$p < .01$	$p < .0001$	$p < .0005$

Table 17. SAIC Rule Set Test: Individual Correlations for Placebo Rules Group

	Graduate	Practitioner 1	Practitioner 2
Graduate			
r_s	-	-	-
Z			
Practitioner 1			
r_s	-.190	-	-
Z	-1.093		
Practitioner 2			
r_s	.446	.149	-
Z	2.562	.858	
Jury			
r_s	.429	.366	.368
Z	2.462	2.100	2.116
significance of Subject/Jury corr.	$p < .01$	$p < .05$	$p < .05$

Table 18. SAIC Rule Set Test: Individual Correlations for No Rules Group

	Graduate	Practitioner 1	Practitioner 2
Graduate			
r_s	-	-	-
Z			
Practitioner 1			
r_s	.682	-	-
Z	3.919		
Practitioner 2			
r_s	.421	.607	-
Z	2.420	3.487	
Jury			
r_s	.622	.601	.606
Z	3.573	3.452	3.478
significance of Subject/Jury corr.	$p < .0005$	$p < .0005$	$p < .0005$

Table 19. SAIC Rule Set Test: Correlations Between Test Groups and Jury

	Locally-Derived Rules	Generic Rules	Placebo Rules	No Rules
r_s	.647	.693	.580	.710
Z	3.719	3.979	3.330	4.081
significance	$p < .0001$	$p < .0001$	$p < .0005$	$p < .0001$

Test of Validity for KUB Rule Set

The KUB Information Systems Department was the field site in which the KUB rule set was developed. The KUB rule set is given in Appendix G.

In the test of the KUB rule set, 36 modules from the KUB inventory of 2181 COBOL batch and transaction processing programs were ranked by 12 subjects. Each of four test groups had three subjects. All subjects were industrial practitioners with at least two years of experience.

The "placebo" rules used in the SAIC test (in Appendix E) were again used in the KUB test.

The ranks assigned by individual subjects and the average ranks for groups are given in Table 20. The correlations between the ranks assigned by individuals within a group and between individuals and the jury are given in Tables 21 through 24. The correlations between average ranks assigned by groups and ranks assigned by the jury are in Table 25.

Findings. Both construct and criterion validity were supported in the KUB test. All group correlations with the jury were statistically significant, but the correlation between the local rules group and the jury was the highest. Ranks of the local rules group correlated with those of the jury at $p < .001$, while ranks of all other groups correlated with the jury's ranks at $p < .05$.

Table 20. Rank Ordering of KUB Modules: Raw Data

Jury Order	Local Rules				Generic Rules				Placebo Rules				No Rules			
	P	P	P	Avg. Rank	P	P	P	Avg. Rank	P	P	P	Avg. Rank	P	P	P	Avg. Rank
1	18	34	15	22.3	23	5	10	27	14.0	9	23	24	35	27.3	33	10
2	12	26	1	13.0	13	1	24	19	14.7	12	1	2	26	9.7	5	10
3	2	1	32	11.7	11	23	3	35	20.3	23	13	29	2	14.7	10	5
4	13	18	22	17.7	17	14	7	17	12.7	6	3	8	24	11.7	6	14
5	19	3	8	10.0	8	2	29	26	19.0	21	14	23	34	23.7	28	7
6	14	19	25	19.3	19	16	6	16	12.7	6	9	7	29	15.0	11	19
7	10	17	23	16.7	16	33	20	30	27.7	35	2	9	27	12.7	8	17
8	35	29	10	24.7	28	34	16	7	19.0	21	21	1	4	8.7	2	2
9	8	12	7	9.0	5	20	5	20	15.0	14	8	13	6	9.0	4	30
10	1	20	2	7.7	2	35	13	25	24.3	29	27	3	19	16.3	16	29
11	7	6	11	8.0	3	6	17	8	10.3	3	16	27	10	17.7	19	8
12	11	16	3	10.0	8	4	26	15	15.0	14	6	10	32	16.0	14	6
13	28	23	24	25.0	29	22	14	21	19.0	21	20	18	20	19.3	22	24
14	15	8	6	9.7	7	13	15	13	13.7	8	33	19	23	25.0	31	27
15	9	15	14	12.7	12	27	19	5	17.0	17	11	11	16	12.7	8	35
16	22	31	20	24.3	26	30	8	31	23.0	27	30	33	18	27.0	32	16
17	16	5	5	8.7	4	15	36	3	18.0	18	22	35	14	23.7	28	23
18	29	13	17	19.7	20	26	9	1	12.0	4	31	31	7	23.0	26	11
19	3	9	16	9.3	6	19	2	23	14.7	12	12	6	9	9.0	4	15
20	17	24	28	23.0	24	29	18	9	18.7	19	19	28	3	16.7	18	24
21	27	2	27	18.7	18	17	4	2	7.7	2	32	21	12	21.7	25	4
22	25	7	29	20.3	21	32	32	18	27.3	33	25	20	5	16.7	18	31
23	20	4	18	14.0	14	18	11	10	13.0	7	17	32	25	24.7	30	14
24	4	27	12	14.3	15	9	1	4	4.7	1	5	14	1	6.7	1	3
25	5	10	4	6.3	1	25	12	6	14.3	10	7	17	36	20.0	24	36
26	6	14	13	11.0	10	31	12	12	21.7	25	29	12	17	19.3	22	4
27	36	21	19	25.3	30	12	35	28	25.0	31	34	30	31	31.7	35	22
28	30	11	21	20.7	22	11	23	34	22.7	26	4	15	30	16.3	16	25
29	33	25	26	28.0	31	28	21	33	27.3	33	10	16	21	15.7	12	29
30	23	36	31	30.0	32	7	31	36	24.7	30	26	26	22	24.7	30	8
31	26	35	33	31.3	34	21	30	11	20.7	24	24	22	11	19.0	20	33
32	24	33	34	30.3	33	3	25	22	16.7	16	15	25	8	16.0	14	21
33	31	30	35	32.0	35	10	33	29	24.0	28	35	34	15	28.0	34	20
34	21	22	30	24.3	26	8	27	14	16.3	15	18	5	13	12.0	7	27
35	32	28	9	23.0	24	36	28	32	32.0	36	28	4	28	20.0	24	32
36	34	32	36	34.0	36	24	34	24	27.3	33	36	36	33	35.0	36	27

Table 21. KUB Rule Set Test: Individual Correlations for Local Rules Group

	Practitioner 1	Practitioner 2	Practitioner 3
Practitioner 1			
r_s	-	-	-
Z			
Practitioner 2			
r_s	.392	-	-
Z	2.318		
Practitioner 3			
r_s	.446	.313	-
Z	2.637	1.850	
Jury			
r_s	.494	.316	.453
Z	2.922	1.872	2.679
significance of Subject/Jury corr.	$p < .005$	$p < .05$	$p < .005$

Table 22. KUB Rule Set Test: Individual Correlations for Generic Rules Group

	Practitioner 1	Practitioner 2	Practitioner 3
Practitioner 1			
r_s	-	-	-
Z			
Practitioner 2			
r_s	-.138	-	-
Z	-.818		
Practitioner 3			
r_s	-.020	.214	-
Z	-.120	1.265	
Jury			
r_s	.085	.484	.065
Z	.503	2.863	.384
significance of Subject/Jury corr.		$p < .005$	

Table 23. KUB Rule Set Test: Individual Correlations for Placebo Rules Group

	Practitioner 1	Practitioner 2	Practitioner 3
Practitioner 1			
r_s	-	-	-
Z			
Practitioner 2			
r_s	.496	-	-
Z	2.936		
Practitioner 3			
r_s	-.073	-.057	-
Z	-.432	-.340	
Jury			
r_s	.408	.260	-.045
Z	2.417	1.541	-.263
significance of Subject/Jury corr.	$p < .01$		

Table 24. KUB Rule Set Test: Individual Correlations for No Rules Group

	Practitioner 1	Practitioner 2	Practitioner 3
Practitioner 1			
r_s	-	-	-
Z			
Practitioner 2			
r_s	.030	-	-
Z	.180		
Practitioner 3			
r_s	.034	.329	-
Z	.201	1.948	
Jury			
r_s	.015	.323	.290
Z	.088	1.910	1.715
significance of Subject/Jury corr.		$p < .05$	$p < .05$

Table 25. KUB Rule Set Test: Correlations Between Test Groups and Jury

	Locally-Derived Rules	Generic Rules	Placebo Rules	No Rules
r_s	.526	.366	.314	.334
Z	3.113	2.165	1.857	1.974
significance	$p < .001$	$p < .05$	$p < .05$	$p < .05$

Reliability of New Process for Deriving Rule Sets

Both construct validity (i.e., a higher correlation between the local rules group and the jury than between any other group and the jury) and criterion validity (i.e., a statistically significant correlation between the local rules group and the jury) were supported in the test of the KUB rule set. Criterion validity was supported in the test of the SAIC rule set, but construct validity was not supported due to a single outlier (i.e., a single individual's very low correlation with the jury).

Ironically, the subjects in the SAIC local rules group other than the outlier had higher correlations with the jury (.744 and .659) than any other subject in any other group. This fact, together with feedback from the outlier (who had a correlation of .102 with the jury) concerning her idiosyncratic application of the rules, suggests that the experimental effect (i.e., the differential effectiveness of the local rule set) was generally in keeping with results in other tests.

As had been the case in the OIS and CTI tests, the relative helpfulness of nonlocal rules differed in the SAIC and KUB tests. In the SAIC test, the relative helpfulness of nonlocal rules appeared to be as follows (in descending order): no rules, generic rules, placebo rules. In the KUB test, however, the relative helpfulness of nonlocal rules was as

follows: generic rules, no rules, placebo rules. While placebo rules were consistently least useful (as might be expected), the effects of generic rules and no rules were not consistent.

Reliability and Validity Across Tests

The aggregation of data from all tests supports a finding of both construct validity and criterion validity of local rules derived through the process proposed by Poore, applied by Binder, and refined in this study.

Reliability of Construct Validity Across Tests. In three of four tests, the experimental group (i.e., the group using locally-derived software quality rules) produced a ranking of modules that correlated more highly with the rank ordering by the organization's jury of practitioners than any of the control groups (i.e., the groups using generic, placebo, or no rules). In the one test in which the local rules group did not correlate more highly with the jury than all other groups, a single individual's results brought the local rules group average down to a level that was numerically below but statistically the same as (i.e., within a confidence interval of) two other groups. Without this extreme individual outlier, the results in this test would have been consistent with the others.

The effect of the experimental condition---the greater ability to make "correct" judgments about the quality of local software by using a locally-derived rule set---does appear to have consistently validated the characterization of the construct under study---"locally-derived rules"---by differentiating its effect from the effects of the control conditions.

Reliability and Criterion Validity Across Tests. In all four tests, the group using locally-derived rules produced a ranking of modules that correlated significantly with the known valid criterion, i.e., the rank ordering by the organization's jury. The correlations between the groups using local rules and the relevant juries were $p < .0001$, $p < .05$, $p < .0001$, and $p < .001$ in the OIS, CTI, SAIC, and KUB tests, respectively.

Aggregate Validity. Figure 7 depicts all individual correlations with relevant juries. The four vertically connected sets of points in each group are correlations for participants in the OIS, CTI, SAIC, and KUB tests respectively. The scale at left indicates the range of correlations that were obtained.

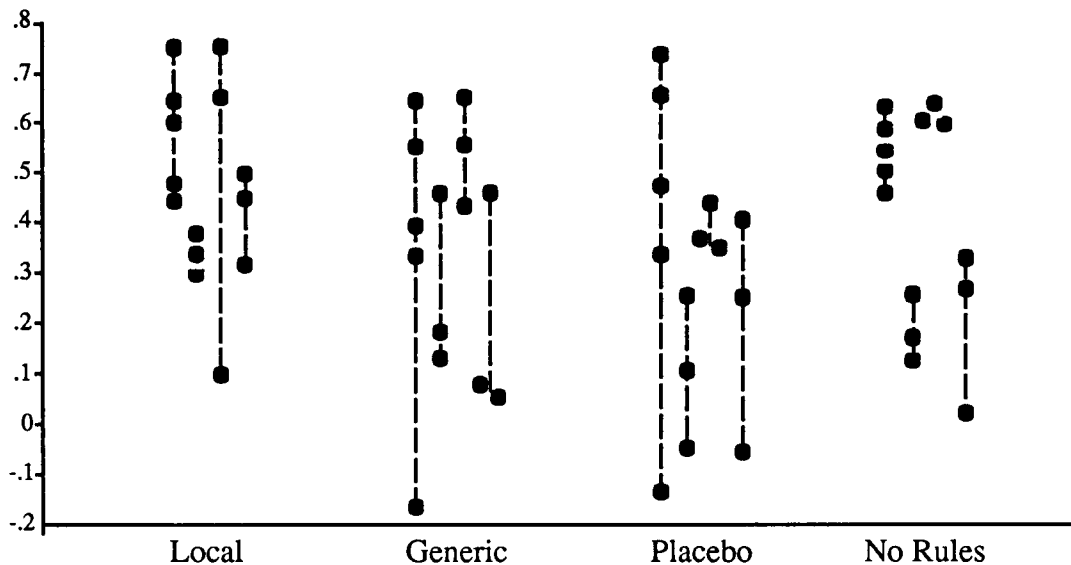


Figure 7. Individual Correlations with Juries

The four vertically connected sets of points for each group represent (in order) OIS, CTI, SAIC, and KUB results.

As is apparent in Figure 7, individuals using locally-derived rules had a higher average correlation with the jury than any other group. The local rules group had a lower standard deviation as well, indicating that not only was the group's average alignment with the jury greater than other groups but more individual members of the group were in alignment with the jury. Table 26 shows the average correlation with the jury, the standard deviation among correlations, and the statistical significance of the average correlation for each group.

Table 26. Average Individual Correlation with the Jury for All Subjects in All Groups

	Locally-Derived Rules	Generic Rules	Placebo Rules	No Rules
r_s	.480	.342	.318	.418
SD	.186	.251	.231	.211
Z	2.839	2.024	1.881	2.473
significance	$p < .005$	$p < .05$	$p < .05$	$p < .01$

It is well known in social science that the average judgment of groups is better than the average judgment of individuals. This fact is reflected in Table 27, which shows the average of group judgments. A comparison of Tables 26 and 27 shows that the average of the group judgments was considerably better than the average judgment of the individuals in the groups. (This finding is not immediately germane to the present study, but is mentioned to reinforce the usefulness of group approaches in software arena as elsewhere.)

Table 27. Average Group Correlation with the Jury for All Groups in All Tests

	Locally-Derived Rules	Generic Rules	Placebo Rules	No Rules
r_s	.579	.479	.446	.483
SD	.173	.174	.245	.253
Z	3.425	2.834	2.639	2.858
significance	$p < .0005$	$p < .005$	$p < .005$	$p < .005$

Aggregate Reliability. In addition to examining reliability of findings across tests, interrater reliability may be examined in the aggregate. Figure 8 depicts all intercorrelations, i.e., correlations between subjects in the same test groups.

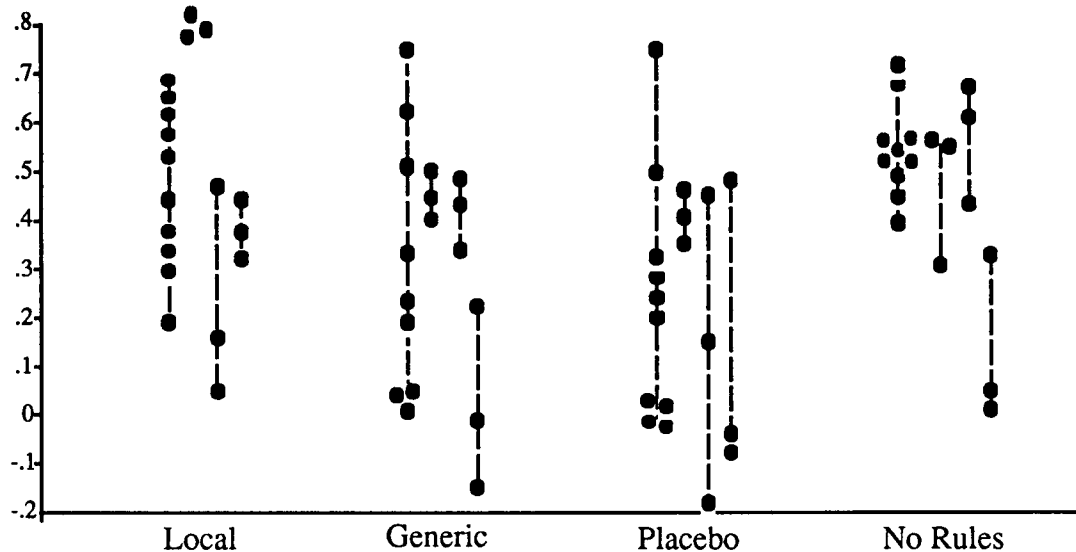


Figure 8. Intercorrelations Between Subjects in the Same Test Groups

The four vertically connected sets of points for each group represent (in order)
OIS, CTI, SAIC, and KUB results.

As can be inferred from Figure 8, individuals using either locally-derived rules or no rules had higher average intercorrelations than individuals in the generic or placebo groups. It appears that nonlocal rules (such as might be imposed by a corporate office or taken from a textbook) are more likely to produce variability in judgment than either local rules or personal judgment alone. Table 28 shows the average intercorrelation, the standard deviation among intercorrelations, and the statistical significance of the average intercorrelation for each group.

Table 28. Average Interrelation Between Subjects

	Locally-Derived Rules	Generic Rules	Placebo Rules	No Rules
r_s	.472	.277	.240	.474
SD	.216	.253	.243	.191
Z	2.792	1.639	1.419	2.804
significance	$p < .005$			$p < .005$

Key Findings. The most important findings in this research were as follows:

- In three of four tests, construct validity was supported (i.e., the experimental group using "locally-derived" software quality rules produced a rank ordering of modules that correlated more highly with the jury than any of the control groups). In the test with the inconsistent outcome, a single individual outlier brought the group average down and accounted for the anomalous result.
- In four of four tests, criterion validity was supported (i.e., there was a statistically significant correlation between the experimental group and the "criterion"---the ranking by the organization's jury of practitioners).
- The aggregated data showed that locally-derived rules were significantly more effective than generic rules, placebo rules, or no rules in enabling nonorganizational programmers to simulate the software quality judgment of experienced practitioners in a given organization.
- The aggregated data showed that use of nonlocal rules produces greater variability in software quality judgment than use of either local rules or no rules.

CHAPTER V

SUMMARY AND CONCLUSIONS

*To achieve true consensus on group goals, there is no better method than having the group set the goals itself. For one thing, participating in setting goals ensures that the goals will be more clearly understood. For another, it gives each group member a chance to commit himself publicly to the group's goal, and this type of public commitment...has been shown to enhance goal acceptance. But participation itself, quite independent of these other factors, seems to be an important determinant in whether an individual truly accepts the working goals of the team with the pursuant increase in productivity.*²²

G.M. Weinberg

There is no single development, in either technology or in management technique, that by itself promises even one order-of-magnitude improvement in productivity, in reliability, in simplicity...

Although we see no startling breakthroughs---and indeed, I believe such to be inconsistent with the nature of software---many encouraging innovations are underway. A disciplined, consistent effort to develop, propagate, and exploit these innovations should indeed yield an order-of-magnitude improvement. There is no royal road, but there is a road....

*[B]uilding software will always be hard. There is inherently no silver bullet.*²³

F. P. Brooks

As Weinberg pointed out long before we were compelled to learn from the Japanese, one of the keys to accomplishing goals is participation in goal setting by the people who must meet the goals. And, as Brooks implies, participation in goal setting is not a silver bullet; it cannot be expected to improve the quality of software by an order of magnitude.

²² G.M. Weinberg, The Psychology of Computer Programming (New York: Van Nostrand Reinhold Co., Inc., 1971), p. 76.

²³ Frederick P. Brooks, Jr., "No Silver Bullet: Essence and Accidents in Software Engineering," *Computer*, April 1987, p. 10-11.

The results of this research bear out both points. Locally-derived software quality rules did enable nonorganizational programmers to make judgments about local software quality that were significantly better than nonorganizational programmers who did not have locally-derived rules to guide their thinking. Though the judgment of programmers using local rules was significantly better from a statistical standpoint, however, it was only incrementally better from a practical standpoint. Locally-derived rules are one of the encouraging innovations to which Brooks alludes, but they are no silver bullet.

Reflections on Field Applications of the Group Process

The focus of this investigation was the validity of rule sets produced by a participative group process, and the reliability of the process in producing valid rule sets. The results of the formal experiments designed to investigate those issues were reported in Chapter IV. Some reflections on the field applications of the process itself, however, may be of value to others who might use it to define local software quality.

The Evolution of the Group Process. In the initial article in this stream of research (Poore, 1988), a group process was proposed through which a locally-appropriate software quality metric and rule set could be derived. The process entailed a "jury of local experts" from an organization's programming staff ranking a random sample of the organization's modules in order of quality, stating the reasons for their decisions as they ranked the modules. Subsequent syntactical analysis produced the local quality metric, and a summary of the jury's reasons for ranking decisions became the rule set.

Binder's research employed Poore's prescription of the process. The focus of Binder's work was the statistical process for deriving the metric once the group work had been completed. The derivation of the rule set---of secondary interest to Binder---was the focus of the present study.

Field applications of the group process in this study included all elements proposed

by Poore and used by Binder, plus three new elements: (1) an individual evaluation of modules prior to the group process, (2) an algorithm for group ranking using the pattern of individual agreements as a basis, and (3) an iterative refinement of the rule set in three separate examinations. These enhancements to the process were intended to maximize individual participation, improve the efficiency of the group's activity, and facilitate consensus. A comparison of experiences by facilitators of the process to date indicates that the enhanced process did, indeed, produce easier consensus, greater consensus, and greater intention to use the results of the process than earlier field applications.

Field Experience with the Enhanced Process. At the outset of each field application, the appointed jurors were somewhat wary about both the purpose of the process and the process itself. Some anticipated acrimonious proceedings upon hearing that group decisions would be made by consensus. Some anticipated being "railroaded" into agreeing to standards that would eliminate all creativity from work. There was fear of hurting other jurors' feelings if and when modules they had authored were reviewed, and fear of having one's own work scrutinized as well.

Three explicit strategies were used to minimize the likelihood that such problems would arise: (1) anonymity of module authorship and individual juror evaluations, (2) beginning with agreements in group ranking, and (3) arbitration of differences based on interests rather than positions.

The purpose of anonymity of module authorship (i.e., blanking out the author's name in source documents) was more to set a tone than to actually ensure objectivity. The jurors were all persons with some tenure in the organization, and often knew who had written the modules. The absence of the author's name, however, was a concrete reflection of the fact that the task at hand was to evaluate practices, not persons, and this attempt to ensure a focus on the work rather than the workers was clearly appreciated by the jurors. A second form of anonymity was used in the process in the initial individual evaluation of modules

by jurors. The feedback report describing agreement patterns among jurors identified the evaluators as "Juror 1," "Juror 2," etc. Some jurors voluntarily identified their evaluations for other jurors, but doing so was their choice. Again, the purpose of anonymity was to avoid personalizing the process and instead focus on module content.

Building on agreements is another aspect of the process that was designed to minimize problems that might arise from differences in opinion. Preexisting agreements between jurors (identified from the initial individual evaluation of modules) were used as the starting point for the group ranking work. The early group experience, therefore, involved decisions on which there was easy consensus, which had an immunizing effect with respect to later discussion of disagreements. The group's early success experience alleviated fears, engendered confidence in the process, and resulted in momentum that sustained the group when the thorniest problems were addressed.

Finally, the facilitator used an approach to mediation of apparent conflict that is a standard technique in arbitration: focusing on *interests* rather than *positions*. When one juror, for example, thought that documentary comments should be in CAPS and source code in mixed case, and another juror thought just the reverse, the two were ultimately able to agree that their common concern was that comments be clearly distinguishable from code. Jurors who seemed to be advocating different practices were usually able to agree that they had a common purpose. The facilitator must not take sides in disagreements between jurors; instead, the facilitator must assist in clarifying the issues by asking questions, offering restatements, and helping jurors give expression to agreements that are as specific as possible but as abstract as necessary. The purpose of the process, after all, is to identify the *existing* sense of software quality in the organization; if no consensus exists on a given point, the point is not properly a part of the rule set.

As the ranking process proceeded and jurors discussed concrete examples of good and bad practices, the shared experience of admiring some modules and identifying specific

improvements needed in others led to an increasingly clear explication of their sense of software quality. By the end, the jurors were convinced that the time had been well spent, and they were quite anxious to distribute the new rule set to their peers for review, feedback, and ultimately, implementation.

The tests of validity for rule sets produced through this group process involved a subject population that had nothing whatsoever to do with the organizations in which the rule sets were derived. This approach---having nonlocal programmers use local rules to evaluate local modules---made for a tough test of the understandability of the rule sets, but provides no information about the effect of the process and the resulting rule set on the organizations in question. The following observation of the facilitator is offered, therefore, to complete that part of the story.

The jurors in these field applications were skeptical at the outset, somewhat begrudging the time involved and the imposition of yet another collateral duty. They emerged from the process, however, with heightened interest in software quality and enthusiasm for involving the entire work group in practical use of the rule set (which they called "standards" or "guidelines"). If the results of these field applications are any indication, this process for defining local software quality is an "encouraging innovation" indeed.

Sources of Variability in Experimental Use of Rule Sets

In each of four validation experiments reported in Chapter IV (i.e., the OIS, CTI, SAIC, and KUB experiments), a correlation was given for the average module ranking by each of four test groups (i.e., the local, generic, placebo, and no rules groups) and the module ranking by the organization's jury of practitioners. These correlations, reported in separate tables in Chapter IV, are summarized in Table 29 for ease of reference.

Variability in the group correlations was manifest in several ways. The range in

group correlations in any given experiment was around .2; the minimum range was .130, and the maximum range was .232. The range in group correlations across experiments, however, was .6; the minimum group correlation was .175, and the maximum group correlation was .775.

Table 29. Summary of All Correlations Between Test Groups and Juries

	Locally-Derived Rules	Generic Rules	Placebo Rules	No Rules
OIS Test	.775 $p < .0001$	.543 $p < .001$	.713 $p < .0001$	.683 $p < .0001$
CTI Test	.369 $p < .05$	.312 $p < .05$	.175	.203
SAIC Test	.647 $p < .0001$	.693 $p < .0001$	.580 $p < .0005$	.710 $p < .0001$
KUB Test	.526 $p < .001$	.366 $p < .05$	.314 $p < .05$	.334 $p < .05$
All Tests	.579 $p < .0005$	.479 $p < .005$	.446 $p < .005$	.483 $p < .005$

The variability among group correlations *within* an experiment is not surprising; an absence of variability among the experimental conditions would have been surprising. What accounts for the variability, however, *between* experiments? Why, for example, are *all* group correlations low in the CTI experiment, low to mid-range in the KUB experiment, and high in the OIS and SAIC experiments?

There are many plausible reasons for the observed variability, some having to do with the differences in modules sets used the experiments and some with conditions under which the research task was performed.

Range in Module Age. CTI is a company that has been in existence less than ten years. Its FORTRAN modules are short and generally structured. OIS, in contrast, is an organization with many old COBOL programs as well as programs written in recent years. The variation in age between CTI and OIS programs, therefore, made quality judgments about a sample of CTI modules inherently more difficult than judgments about OIS modules. The similarity between CTI modules required "close calls" in the ranking, and this difficulty in ranking decisions was reflected in low group correlations with the jury. The greater differences in quality among OIS modules, on the other hand, made for easier ranking decisions and consequently higher group correlations with the jury.

Similarity Between Jurors and Test Subjects. The SAIC jurors were all trained computing professionals, and most had advanced degrees. The KUB jury was a more mixed group; two jurors had undergraduate degrees in computer science (one recent and one more than ten years old) and two were practitioners whose expertise had been developed on the job. The test subjects in both the SAIC and KUB experiments were professionals whose backgrounds were more like the SAIC jurors than the KUB jurors. It is not surprising, then, that the similarity in judgment (i.e., the size of the correlation) between test subjects and jurors was greater in the SAIC experiment than the KUB experiment.

Test Conditions. The logistics of applied research sometimes necessitate sacrifices in experimental control. In this study, subjects in the validation experiments performed the research task under a variety of conditions. In the OIS experiment, subjects were in a supervised setting and had a fixed amount of time to perform the research task. In the KUB experiment, subjects received both oral and written instructions about the research task, but actually performed the task at a place and time of their own choosing. The higher correlation in the OIS experiment than the KUB experiment could be related to a higher average time-on-task by subjects.

Individual Variability. Other factors influencing variability among group correlations surely stem from oddities in individual performance. Figure 6 in Chapter IV shows numerous outliers among individual correlations with the juries. Insufficient time-on-task, excessive time-on-task, individualistic applications of rules, and other aberrations in individual performance led to extreme results in some cases. Since the test groups were small (5 in each OIS test group and 3 in all other test groups), extreme individual results had a considerable impact on group averages.

While the observed variability in the data raises questions about the confidence with which conclusions may be drawn, it is probably a reasonable reflection of the variability in use of rule sets that would occur in practice. The very individual and organizational factors that produced variability in experimental results will exist in any applied setting as well.

Using Rule Sets in Practice

Poore's original concept for the rule set was as an operational definition of software quality for a given working environment. A proper "operational definition" specifies the actual operations that must be performed to achieve a particular state. Such specificity with respect to software quality implies several uses for a rule set in an organization.

First and foremost, a rule set defines the standard practices through which the work group will achieve the highest level of quality of which it is capable at a given point in time. Derived from local experts' review of a representative set of local modules, the rules reflect the best practices currently in use in the organization. They define a level of quality that *has* been achieved in part, and that the group may aspire to achieve in full.

Standards are abstractions unless they become institutionalized through concrete implementation. Ways in which the rule set might be used in everyday practice are as

- the basis for an expert system tool used in local software development and maintenance,
- a framework for code walkthroughs,

- an aid in acculturation of new employees, and
- a reference in performance counseling.

Finally, the rule set can be a working prescription for redefinition of and improvement in software quality over time. As the technical environment changes, the rule set can be revised to reflect practices that are appropriate for the prevailing context. And as capability increases, the rule set can be strengthened to reflect higher quality goals.

Continuation of This Stream of Research

Efforts to replicate the findings reported in this research with larger subject groups would be instructive. In particular, use of the same experimental design with test subjects from the local organization (rather than nonorganizational programmers as were used in this study) may provide evidence of more immediate organizational benefits of local rule sets.

The jury's module measurement process is the basis for subsequent analyses. Poore originally prescribed that interval rather than ordinal data be generated by the jury. The pair of ranked modules that were closest in quality were to define the unit distance, and the distances between other modules were to be determined on that basis. Binder's juries did generate interval data by this method, but the group process was conflictual and the group consensus weak. Simple ranking was used in this study, and the group process was smooth and the group consensus strong. The relationship of the difference in measurement approach to the group experience is unclear, and is an area for further study.

Continued work on the development of the rule set would be useful. The efficiency of the process for deriving rule sets in this study was such that deliberations were limited to content, internal consistency, and organization. Issues such as weighting and hierarchy were not extensively addressed. Also, test subjects in this study were not debriefed concerning the understandability of the local rule set. Feedback from people who try to use the rule set would surely point to ways to improve its structure.

Research that would relate local rules and local metrics would indicate the consistency between the qualitative and quantitative tracks being pursued in this stream of research. Given Binder's technique for deriving a valid local metric and the process used in this study for deriving a valid rule set, an obvious additional step is examination of the correlation between rank orders produced by the metric and the rule set. If such correlation is "good," the usefulness of the rule set in increasing the metric value could be examined. The rule set could be used by an experimental group in a program maintenance task, not used by a control group performing the same maintenance task, and the relative change in the metric value for each group observed.

Generalization of the local rule set beyond its setting is not intended. It would be interesting, however, to observe similarities and differences in rule sets across settings. Variations across computing environments, applications, corporate cultures, and practitioner profiles could be considered after sufficient field application at the local level.

The fullest consideration of software quality must, ultimately, extend local measures beyond programming practices to the entire software development process. The existence of source code is common to all software environments, and provides a ready starting point for any organization that wishes to consider software quality. It is the easiest place to begin. The quality of source code, however, is largely determined by process of its development--- how it was specified, designed, and certified. A complete characterization of local software quality must ultimately include local measures of the entire software development process.

BIBLIOGRAPHY

BIBLIOGRAPHY

- Adelson, B. "Problem Solving and the Development of Abstract Categories in Programming Languages." *Memory and Cognition*, 9 (1981), pp. 422-433.
- Binder, L.H. *Local Quality Metric for ACOMS FORTRAN Programs*. University of Tennessee Technical Report, April 1988a.
- Binder, L.H. *Local Quality Metric for CTD COBOL Programs*. University of Tennessee Technical Report, April 1988b.
- Binder, L.H. *Local Quality Metric for Real-Time FORTRAN Subroutines*. University of Tennessee Technical Report, April 1988c.
- Binder, L.H. and J.H. Poore. "Field Experiments with Local Software Quality Metrics." *Software---Practice and Experience*, July 1990.
- Binder, L.H., J.H. Poore and T.J. Hsiao. *A Software Quality Circle Experiment for a COBOL Batch Environment*. University of Tennessee Technical Report, April 1987.
- Bouchard, T.J. "Personality, Problem Solving Procedure, and Performance in Small Groups." *Journal of Applied Psychology Monograph* 53, 1969.
- Brooks, Jr., Frederick P. "No Silver Bullet: Essence and Accidents in Software Engineering." *Computer*, April 1987, pp. 10-19.
- Brunswik, E. "Organismic Achievement and Environmental Probability." *Psychological Review*, 50 (1943), pp. 255-272.
- Brunswik, E. "Representative Design and Probabilistic Theory in a Functional Psychology." *Psychological Review*, 62 (1955), pp. 193-217.
- Couger, J.D. "Circular Solutions." *Datamation*, January 1983, pp. 135-142.
- Delbecq, A.L., A.H. Van de Ven, and D. H. Gustafson. *Group Techniques for Program Planning*. Glenview, Il.: Scott, Foresman, 1975.
- Grady, Robert B. and Deborah Caswell. *Software Metrics: Installing a Company-Wide Program*. Englewood Cliffs, N.J.: Prentice Hall, Inc., 1987
- Gryna, F.M. *Quality Circles: A Team Approach to Problem Solving*. New York: AMACOM, 1981.
- Hammond, K.R., J. Rohrbaugh, J. Mumpower, and L. Adelman. "Social Judgment Theory: Applications in Policy Formation," in M.F. Kaplan and S. Schwartz (Eds.), *Human Judgment and Decision Processes in Applied Settings*. New York: Academic Press, 1977.

- Hammond, K.R., T.R. Stewart, B. Brehmer, and D. Steinmann. "Social Judgment Theory," in M.F. Kaplan and S. Schwartz (Eds.), *Human Judgment and Decision Processes: Formal and Mathematical Approaches*. New York: Academic Press, 1975.
- Holloman, C.R., and H.W. Hendrick. "Adequacy of Group Decisions As a Function of the Decision Making Process." *Academy of Management Journal* 15 (1972), pp. 175-184.
- Janis, I. L. *Victims of Groupthink*. Boston: Houghton-Mifflin, 1972.
- Kellogg, D.L. "The Implications of Japanese Software Factories and Project SIGMA Within Software Development Environments" in *Proceedings of the IEEE/AIAA 7th Digital Avionics Systems Conference*, 1986, pp. 28-32.
- Kernighan, B., and P. Plauger. *The Elements of Programming Style*. New York: McGraw Hill Co., 1978.
- Lawler, III, E.E. and S.A. Mohrman. "Quality Circles After the Fad." *Harvard Business Review*, January-February 1985, pp. 65-71.
- Leedy, P.D. *Practical Research: Planning and Design*. New York: MacMillan Publishing Company, 1985.
- Levering, R. *The 100 Best Companies to Work for in America*. New York: Addison-Wesley, 1984.
- Leventhal, L.M. *Perception of Software Quality: Aesthetics and Expertise*. University of Michigan Doctoral Dissertation, 1987.
- Linstone, H.A. and M. Turoff. *The Delphi Method: Techniques and Applications*. Reading, Ma.: Addison-Wesley, 1975.
- Molzberger, P. "Aesthetics and Programming" in *Proceedings CHI'83, Human Factors in Computing Systems* (pp. 247-250). New York: ACM, 1984.
- Mizuno, Y. "Software Quality Improvement." *Computer*, March 1983, pp. 66-72.
- Ott, L. *An Introduction to Statistical Methods and Data Analysis*. New York: PWS-KENT Publishing Company, 1988.
- Ouchi, W. *Theory Z: How American Business Can Meet the Japanese Challenge*. Reading, Ma.: Addison-Wesley Publishing Company, 1981.
- Peters, J., and R. Waterman. *In Search of Excellence: Lessons from America's Best Run Companies*. New York: Harper and Row, 1982.
- Phillips, R.T. "An Approach to Software Causal Analysis and Defect Extinction" in *GLOBECOM '86: IEEE Global Telecommunications Conference Record* (pp. 412-416). New York: IEEE, 1986.

- Pickler, L. "Quality Circles in the Systems Environment." *Journal of Systems Management*, November 1983, pp. 14-16.
- Poore, J.H. "Derivation of Local Software Quality Metrics." *Software---Practice and Experience*, November 1988, pp. 1017-1027.
- Prell, E.M., and A.P. Sheng. "Building In Quality and Productivity to a Large Software System" in *First International Conference on Computers and Applications* (pp. 233-239). Silver Spring, Md.: IEEE Computing Society Press, 1984.
- Rohrbaugh, J. "Improving the Quality of Group Judgment: Social Judgment Analysis and the Delphi Technique." *Organizational Behavior and Human Performance*, October 1979, pp. 73-92.
- Rohrbaugh, J. "Improving the Quality of Group Judgment: Social Judgment Analysis and the Nominal Group Technique." *Organizational Behavior and Human Performance*, October 1981, pp. 272-288.
- Ryan, J.R. "Software Product Quality Assurance" in *AFIPS Conference Proceedings* (pp. 393-398). Arlington, Va.: AFIPS Press, 1982.
- Shneiderman, B. "Measuring Computer Program Quality and Comprehension." *International Journal of Man-Machine Studies* 9 (1977), pp. 465-478.
- Soloway, E., and K. Ehrlich. "Empirical Studies of Programming Knowledge." *IEEE Transactions on Software Engineering*, September 1984, pp. 595-609.
- Steiner, I. D. *Group Process and Productivity*. New York: Academic Press, 1971.
- Sullivan, J.J. *An Experimental Study of a Method for Improving the Effectiveness of the Nominal Group Technique*. University of Florida Doctoral Dissertation, 1978.
- Thompson, J., and A. Tuden. "Strategies, Structures, and Processes of Organizational Decision Making," in James D. Thompson (ed.), *Comparative Studies in Administration*. Pittsburgh: University of Pittsburgh Press, 1959.
- Tolman, E.C. *Purposive Behavior in Animals and Men*. New York: Century, 1932.
- Tolman, E.C. and E. Brunswik. "The Organism and the Causal Texture of the Environment." *Psychological Review*, 42 (1935), pp. 43-77.
- Van de Ven, A.H. and A. Delbecq. "The Effectiveness of Nominal, Delphi, and Interacting Group Decision Making Processes." *Academy of Management Journal*, 17 (1974), pp. 605-632.
- Weinberg, Gerald M. *The Psychology of Computer Programming*. New York: Van Nostrand Reinhold Co., Inc., 1971.

APPENDICES

Appendix A. OIS Software Quality Rule Set

Rule Group 1 - Weight 4

1. Implement one function per module.
2. Document each module.
3. Avoid GO TO. Use only when necessary.
4. Never use ALTER statement.

Rule Group 2 - Weight 3

5. Use consistent procedure naming convention.
6. Use two procedure names per module, one entry and one exit.
7. Each module should have clearly marked, single-entry, single-exit points.
8. Procedure names should have numeric sequencing mechanism in the first node of the name. Other sequencing mechanism OK if necessary.
9. Procedure names should appear on lines by themselves.

Rule Group 3 - Weight 2

10. Use proper indentation rules.
11. Avoid compound conditionals with NOT.
12. Avoid literals in the PROCEDURE DIVISION.
13. Each file should have a single read/write.
14. Check all significant return codes from a CALL.
15. Use simple IF instead of nested IF.
16. Use meaningful data names.

Rule Group 4 - Weight 1

17. Avoid redundant code such as "ELSE NEXT SENTENCE"
18. Each logical statement should end with a period.
19. Use single assignments per statement.
20. Properly comment out old code.
21. Avoid redundant condition checking.
22. Avoid data name qualification.

Appendix B. CTI Software Quality Rule Set

<u>Weight</u>	<u>Rule</u>
336	Include adequate documentation.
128	Modules should be formatted so that they are readable.
117	Avoid GOTO statements.
110	Avoid incorrect or misleading documentation.
97	Avoid use of EXIT within a subprogram.
81	Document the limitations of a module.
74	Avoid use of COMMON.
61	Avoid inefficient coding.
50	Use meaningful variable names.
41	Avoid declaration of unreferenced variables.
38	Avoid unnecessarily complex conditional statements.
37	Document magic numbers (numeric literals) where necessary.
36	Use one exit from a module.
35	Use simple CALL structures for subprograms.
26	Document CALL structures adequately.
22	Use consistent coding practices within a module.
20	Avoid modification of input variables to a subprogram.
17	Document unclear algorithms.
15	Avoid implementation of trivial functions as subprograms.
12	Avoid leaving unused code in a module even if it is commented out.
11	Always use explicitly declared variables.
9	Do not assume the size of an input array for a subprogram.
5	Avoid unreferenced procedure identifiers.
3	Line comments up appropriately.
1	Use consistent indentation.
1	Avoid indentation that is too deep.

Appendix C. Sample Composite Preliminary Evaluation

Composite Preliminary Evaluation

a step in development of SAIC software quality standards

October 19, 1990

Contents

Definitions

Software Quality Definitions: Raw Data

Summary of Software Quality Definitions

Rules

Software Quality Rules: Raw Data

Summary of Software Quality Rules

Modules

Individual Evaluation of Modules: Raw Data

Agreement Pattern

Software Quality Definitions: Raw Data (Individual Definitions)

Software Quality is...

software that is easy to read and maintain. Follows a consistent standard and is documented.

modularly designed software that is easily understandable, well documented, and easy to maintain.

the state of software that clearly and concisely depicts the objective of a module and the procedures used to establish that objective.

a system that supports the production of software that is correct, robust, flexible, maintainable, efficient, and lucid.

<i># mentions</i>	<i>descriptors</i>	<i>theme</i>
3	easy to understand, lucid, clear and concise	Understandability
1	easy to read	
1	follows a consistent standard	
3	easy to maintain	Maintainability
1	flexible	
1	modularly designed	
1		
2	well documented	Documentation
1	depicts the objective of a module and procedures	
1	correct	Correctness
1	robust	
1	depicts...procedures used to establish that [module's] objective	
1	efficient	Efficiency

Summary of Software Quality Definitions

"Software quality" in the SAIC/OSTI environment is defined as software that is correct, well documented, easy to read and understand, easy to maintain, and efficient.

- Software is correct if it meets its stated objective and is robust.
- Software is well-documented if it clearly and concisely depicts the objective of a module and its procedures.
- Software is easy to read and understand if it follows a consistent standard.
- Software is easy to maintain if it is modularly designed and flexible.

Software Quality Rules: Raw Data (Individual Rules)

Importance Level 1

1. Provide a header, explain inputs/outputs, functionality, edit history, what's going on.
2. Make use of DO WHILE, rather than GO TO.
3. Use descriptive variable name, avoid X I J etc.
4. Indent inside of IF - ELSE, DO WHILE. Don't indent just part of it.
5. Try to keep one statement per line, at most.
6. Make use of IF-ELSE IF structure, rather than just IF THEN.
7. Avoid statement labels that are never used.

8. Use well documented, meaningful variable names.
9. Use structured programming, don't jump around.
10. Provide overall description of module and section comments.
11. Use language, system features which are provided to minimize errors. (example---in FORTRAN---can enforce typing of variables)
12. Write header documentation which describes what the module does.
13. Use "data-driven" coding techniques whenever possible.

14. Module should contain documented header information.
15. Repetitive actions involving data should be table driven.
16. Provide in-line comments often and where needed.
17. State the purpose or objective of the module clearly.
18. Define dependent codes and authorities used.
19. Use meaningful variable names.
20. Document rules for validation.
21. Define parameters.

22. Use logical names for area locations on filenames appearing in code.
23. Use IMPLICIT NONE.
24. Use OPTIONS /EXTEND_SOURCE.
25. Have a detailed module header.
26. Use variable names that suggest that variable's usage.
27. Comment (blocks/sequences of) operations inside the module.
28. Use submodules/utility modules where possible and appropriate---make use of Run Time Library routines; don't reinvent.
29. Don't assume good data--do error checking--make module robust--reduce vulnerability

30. Make repetitive operations data-driven rather than hardcoded.
31. Be sure module functions correctly. Test it thoroughly.
32. Declare received character variables in subprograms as passed-length rather than fixed-length.
33. Make required input as general as possible---do not unnecessarily restrict choices for users.
34. In setting sizes on variables to hold unknown input, don't be too skimpy---allow for very peculiar cases and then test input to ensure it matches your assumptions.
35. Inside the code, reuse workspace variables wherever possible.
36. Design your user conventions (return codes, success/failure values, naming conventions, etc.) to be consistent with system conventions wherever possible.
37. If you're writing a function, assign it a value!
38. Do not use constant values for system status values, keystroke codes, condition codes returned by system functions---use the correct system names for these codes and INCLUDE the appropriate system library modules to define the names.
39. Do not simply silently handle error conditions---report weirdness as discovered.
40. Explain variables passed in to a subprogram---meaning, format, and limitations.
41. Design your code for maximum usage with minimum modification and maintenance.

Importance Level 2

42. Include comments in body of program, before block of code.
 43. Use blank lines to separate blocks and logical parts.
 44. Use all CAPS for code, use mixed case for comments.
 45. In declaration of variable, one variable per line and alphabetize.
 46. When maintaining code, follow existing style. If code is all uppercase, don't modify with lowercase.
- ***
47. Use white space and section comments to make easier to understand and maintain.
 48. Use meaningful "loop-controls" and "decision makers".
 49. Define parameters in file and use descriptive names.
 50. Use system routines as much as possible.
 51. Reuse routines as much as possible.
- ***
52. Use tabular indentation techniques to show dependent actions.
 53. Explain acronyms referenced.
 54. Document the use and purpose of logicals.
 55. Document the purpose/use of include files.
 56. Specific references of disk areas should be by logicals.
 57. Module should only have one purpose. (Brevity)
 58. Actions invoked should be from a primary main area.

- 59. Describe briefly the purpose of subprograms called.
- 60. First line should specify name of module.
- ***
- 61. Use DO WHILE / END DO instead of numbers when natural to do so.
- 62. Use (user) parameter names and (user) variable names instead of constants or constant expressions if the constants have meaning.
- 63. Annotate modifications so that modification history is reasonably clear.
- 64. Include instructions for module modification for correct maintenance.
- 65. Use indentation and vertical spacing to visually represent logical blocks of code.
- 66. With comments in the code, document complex operations especially carefully, with indentation as appropriate, to clarify as much as possible.
- 67. If a multi-valued "CASE" statement (IF...ELSE IF..., etc.) is unavoidable, arrange the cases in a natural order, if such an order exists.
- 68. Prefer "IF" plus repeated "ELSE IF" clauses to a series of "IF --- END IF" clauses.
- 69. Factor out repeated "IF" conditions to reduce code volume.
- 70. Do not include unnecessary code that does not serve to clarify. Such code misleads by implying by its presence that it is necessary.

Importance Level 3

- 71. Commenting is good, but avoid over comments and repeating stuff.
- 72. Space consistently. Example: don't space before "(" sometimes and other times use TAB before "(".
- 73. Tabing for indentation is OK unless lots of embedding then lines get too short. Indent with spaces so more will fit on a line.
- 74. Try to use positive comparison over negative. Like EQ instead of NE.
- 75. If a block of code is commented out, give an explanation.
- ***
- 76. Assign input/output unit numbers to variable names.
- ***
- 77. State if module is used as part of a system or is generic in nature.
- ***
- 78. Identify where implicit inputs and outputs reside (what COMMON area, etc.) in documentation.
- 79. Include variable (optional) run-time debugging output.
- 80. Group variables of the same type together in declarations.
- 81. Use statement functions to represent commonly used operations when appropriate.
- 82. Set coding style so your code is easily distinguishable from your comments.
- 83. Don't cramp your intra-line style. Use blanks/white space to separate tokens.
- 84. Prefer single quotes (') to Hollerith (nH) notation.

Summary of Software Quality Rules

Correctness

Importance Level 1

11. Use language, system features which are provided to minimize errors. (example---in FORTRAN---can enforce typing of variables)
29. Don't assume good data---do error checking---make module robust---reduce vulnerability---check arguments.
31. Be sure module functions correctly. Test it thoroughly.
37. If you're writing a function, assign it a value!
39. Do not simply silently handle error conditions---report weirdness as discovered.

Importance Level 2

70. Do not include unnecessary code that does not serve to clarify. Such code misleads by implying by its presence that it is necessary.

Importance Level 3

79. Include variable (optional) run-time debugging output.

Documentation

Importance Level 1

1. Provide a header, explain inputs/outputs, functionality, edit history, what's going on.
10. Provide overall description of module and section comments.
12. Write header documentation which describes what the module does.
14. Module should contain documented header information.
16. Provide in-line comments often and where needed.
17. State the purpose or objective of the module clearly.
18. Define dependent codes and authorities used.
20. Document rules for validation.
21. Define parameters.
25. Have a detailed module header.
27. Comment (blocks/sequences of) operations inside the module.
40. Explain variables passed in to a subprogram---meaning, format, and limitations.

Importance Level 2

42. Include comments in body of program, before block of code.
49. Define parameters in file and use descriptive names.
53. Explain acronyms referenced.
54. Document the use and purpose of logicals.
55. Document the purpose/use of include files.
59. Describe briefly the purpose of subprograms called.

- 63. Annotate modifications so that modification history is reasonably clear.
- 64. Include instructions for module modification for correct maintenance.
- 66. With comments in the code, document complex operations especially carefully, with indentation as appropriate, to clarify as much as possible.

Importance Level 3

- 71. Commenting is good, but avoid over comments and repeating stuff.
- 75. If a block of code is commented out, give an explanation.
- 77. State if module is used as part of a system or is generic in nature.
- 78. Identify where implicit inputs and outputs reside (what COMMON area, etc.) in documentation.

Related Rules:

- 1, 10, 12, 14, 17, 25 (header)
- 10, 27, 42, 66 (comments in body)
- 21, 40, 49 (parameters)

Readability/Understandability

Importance Level 1

- 4. Indent inside of IF - ELSE, DO WHILE. Don't indent just part of it.
- 5. Try to keep one statement per line, at most.
- 7. Avoid statement labels that are never used.
- 8. Use well documented, meaningful variable names.

Importance Level 2

- 43. Use blank lines to separate blocks and logical parts.
- 44. Use all CAPS for code, use mixed case for comments.
- 45. In declaration of variable, one variable per line and alphabetize.
- 46. When maintaining code, follow existing style. If code is all uppercase, don't modify with lowercase.
- 47. Use white space and section comments to make easier to understand and maintain.
- 52. Use tabular indention techniques to show dependent actions.
- 60. First line should specify name of module.
- 65. Use indentation and vertical spacing to visually represent logical blocks of code.

Importance Level 3

- 72. Space consistently. Example: don't space before "(" sometimes and other times use TAB before "(".
- 73. Tabing for indentation is OK unless lots of embedding then lines get too short. Indent with spaces so more will fit on a line.
- 80. Group variables of the same type together in declarations.
- 82. Set coding style so your code is easily distinguishable from your comments.
- 83. Don't cramp your intra-line style. Use blanks/white space to separate tokens.

Related Rules:

- 4, 52, 65 (indentation)
- 43, 47, 65 (blank lines)
- 44, 82 (distinguishing code/comments)

Control Structure

Importance Level 1

- 2. Make use of DO WHILE, rather than GO TO.
- 6. Make use of IF-ELSE IF structure, rather than just IF THEN.
- 9. Use structured programming, don't jump around.

Importance Level 2

- 58. Actions invoked should be from a primary main area.
- 61. Use DO WHILE / END DO instead of numbers when natural to do so.
- 67. If a multi-valued "CASE" statement (IF...ELSE IF..., etc.) is unavoidable, arrange the cases in a natural order, if such an order exists.
- 68. Prefer "IF" plus repeated "ELSE IF" clauses to a series of "IF --- END IF" clauses.
- 69. Factor out repeated "IF" conditions to reduce code volume.

Importance Level 3

- 74. Try to use positive comparison over negative. Like EQ instead of NE.
- 81. Use statement functions to represent commonly used operations when appropriate.

Related Rules:

- 2, 9, 61 (GO TO)
- 6, 68 (use ELSE IF)

Data

Importance Level 1

- 3. Use descriptive variable name, avoid X I J etc.
- 8. Use well documented, meaningful variable names.
- 13. Use "data-driven" coding techniques whenever possible.
- 15. Repetitive actions involving data should be table driven.
- 19. Use meaningful variable names.
- 22. Use logical names for area locations on filenames appearing in code.
- 23. Use IMPLICIT NONE.
- 26. Use variable names that suggest that variable's usage.
- 30. Make repetitive operations data-driven rather than hardcoded.
- 32. Declare received character variables in subprograms as passed-length rather than fixed-length.
- 33. Make required input as general as possible---do not unnecessarily restrict choices for users.
- 34. In setting sizes on variables to hold unknown input, don't be too skimpy---allow for very peculiar

cases and then test input to ensure it matches your assumptions.

- 35. Inside the code, reuse workspace variables wherever possible.
- 36. Design your user conventions (return codes, success/failure values, naming conventions, etc.) to be consistent with system conventions wherever possible.
- 38. Do not use constant values for system status values, keystroke codes, condition codes returned by system functions---use the correct system names for these codes and INCLUDE the appropriate system library modules to define the names.

Importance Level 2

- 48. Use meaningful "loop-controls" and "decision makers".
- 56. Specific references of disk areas should be by logicals.
- 62. Use (user) parameter names and (user) variable names instead of constants or constant expressions if the constants have meaning.

Importance Level 3

- 76. Assign input/output unit numbers to variable names.
- 84. Prefer single quotes (') to Hollerith (nH) notation.

Related Rules:

- 3, 8, 19, 26, 48 (meaningful names)
- 13, 15, 30 (data/table-driven operations)
- 22, 56 (logical names for reference to disk areas)
- 30, 38, 62 (hardcoding)

Modularity

Importance Level 1

- 28. Use submodules/utility modules where possible and appropriate---make use of Run Time Library routines; don't reinvent.

Importance Level 2

- 50. Use system routines as much as possible.
- 51. Reuse routines as much as possible.
- 57. Module should only have one purpose. (Brevity)

Related Rules:

- 28, 50 (system routines)
- 28, 51 (reuse)

Unclassified

Importance Level 1

- 24. Use OPTIONS /EXTEND_SOURCE.
- 41. Design your code for maximum usage with minimum modification and maintenance.

Individual Evaluation of Modules: Raw Data

Modules	Jury Members				Agreement Code *
	1	2	3	4	
1	1	1	2	1	NA ₁ 1
2	4	3	4	4	NA ₁ 4
3	1	2	1	1	NA ₁ 1
4	1	3	2	2	D 2
5	3	4	4	4	NA ₁ 4
6	4	3	4	4	NA ₁ 4
7	2	1	4	3	D
8	3	4	4	4	NA ₁ 4
9	4	4	3	4	NA ₁ 4
10	1	1	1	1	A 1
11	2	3	2	3	NA ₂ 2
12	4	4	4	3	NA ₁ 4
13	2	1	1	2	NA ₂ 1
14	4	4	4	4	A 4
15	1	2	1	2	NA ₂ 1
16	4	4	3	2	D 4
17	1	2	1	1	NA ₁ 1
18	3	2	2	2	NA ₁ 2
19	2	3	4	1	D
20	2	2	3	2	NA ₁ 2
21	3	1	1	2	D 1
22	4	4	4	4	A 4
23	2	1	2	3	D 2
24	2	3	2	3	NA ₂ 2
25	3	1	3	3	NA ₁ 3
26	3	3	3	4	NA ₁ 3
27	4	4	2	3	D 4
28	4	2	3	2	D 2
29	2	1	2	1	NA ₂ 1
30	3	2	3	2	NA ₂ 2
31	2	4	3	4	D 4
32	1	3	3	1	D 1
33	1	2	1	1	NA ₁ 1
34	1	1	1	1	A 1
35	3	3	2	3	NA ₁ 3
36	3	2	1	3	D 3

* Agreement Codes:

- A Agreement
- NA₁ Near Agreement (1 juror differs)
- NA₂ Near Agreement (2/2 split - adjacent cats.)
- D Disagreement
- # Highest Mode

Agreement Pattern

Quality Level	Agreements all jurors agree	Near Agreements 1 jurors differs or 2/2 split in adjacent categories	Disagreements 3 or more jurors disagree; modules listed in order of highest mode
1	10 34	1 3 17 33 13 15 29	21 32
2		18 20 11 24 30	4 23 28
3		25 26 35	36
4	14 22	2 5 6 8 9 12	16 27 31 7 19

Appendix D. Kernighan and Plauger's Rules from *The Elements of Programming Style*

Say what you mean, simply and directly.

Use library functions.

Avoid temporary variables.

Write clearly --- don't sacrifice clarity for "efficiency."

Let the machine do the dirty work.

Replace repetitive expressions by calls to a common function.

Parenthesize to avoid ambiguity.

Choose variable names that won't be confused.

Avoid unnecessary branches.

Use the good features of a language; avoid the bad ones.

Don't use conditional branches as a substitute for a logical expression.

Use DO-END and indenting to delimit groups of statements.

Use IF-ELSE to emphasize that only one of two actions is to be performed.

Use DO and DO-WHILE to emphasize the presence of loops.

Make your programs read from top to bottom.

Use IF...ELSE IF...ELSE IF...ELSE...to implement multi-way branches.

Use the fundamental control flow constructs.

Avoid THEN-IF and null ELSE.

Avoid ELSE GOTO and ELSE RETURN.

Follow each decision as closely as possible with its associated action.

Use data arrays to avoid repetitive control sequences.

Choose a data representation that makes the program simple.

Modularize. Use subroutines.

Make the coupling between modules visible.

Each module should do one thing well.

Make sure every module hides something.

Let the data structure the program.

Use recursive procedures for recursively-defined data structures.

Test input for validity and plausibility.

Make sure input cannot violate the limits of the program.

Terminate input by end-of-file or marker, not by count.

Identify bad input; recover if possible.
Treat end of file conditions in a uniform manner.
Use uniform input formats.
Make input easy to proofread.
Use free-form input when possible.
Use self-identifying input. Allow defaults. Echo both on output.
Localize input and output in subroutines.
Make sure all variables are initialized before use.
Initialize constants with DATA statements or INITIAL attributes;
 initialize variables with executable code.
Watch out for off-by-one errors.
Take care to branch the right way on equality.
Avoid multiple exits from loops.
Make sure your code "does nothing" gracefully.
Test programs at their boundary values.
Program defensively.
Don't compare floating point numbers just for equality.
Make it right before you make it faster.
Keep it right when you make it faster.
Make it clear before you make it faster.
Let your compiler do the simple optimizations.
Don't strain to reuse code; reorganize instead.
Make sure special cases are truly special.
Keep it simple to make it faster.
Don't diddle code to make it faster --- find a better algorithm.
Make sure comments and code agree.
Don't just echo the code with comments --- make every comment count.
Use variable names that mean something.
Use statement labels that mean something.
Format a program to help the reader understand it.
Indent to show the logical structure of a program.
Document your data layouts.
Don't over-comment.

Appendix E. Placebo Rule Set

Design

Be mindful of application dependencies and interoperability considerations.

A module should be as long as necessary to perform its function.

Remember the user.

Declarations

Do not use reserved words as variable names.

Use correct data types and structures.

Variables should be used for values that change; constants should be used for values that do not change.

Initialize variables when appropriate.

Structure and Syntax

Individual lines in a module should have no cross dependencies.

Make proper use of sequence, selection, and iteration structures for program control; avoid unnecessary code.

Nest control structures correctly.

Assignment statements should be executed in the correct order.

Use arithmetic operators for calculations only.

The predicate of an IF statement should be a relational expression that evaluates to true or false.

Use the IF-THEN-ELSE construct for branching conditions only.

Exercise due deliberation with the use of GO TO: do not use GO TO if it is not necessary, but use GO TO when it is necessary.

Use parentheses to override precedence levels.

Use legal syntax. If you are unsure of syntax, consult the manual.

Documentation

Use comments when necessary.

Avoid incorrect documentation.

Appendix F. SAIC Software Quality Rule Set

Software Quality in the SAIC/OSTI VAX FORTRAN environment is defined as software that is correct, well documented, easy to maintain, easy to read and understand, and efficient.

- Software is correct if it meets its stated objective and is robust.
- Software is well-documented if it clearly and concisely depicts the objective of a module and its procedures.
- Software is easy to maintain if it is modularly designed and flexible.
- Software is easy to read and understand if it follows a consistent standard.

Software quality is supported by the following set of rules.

Correctness

Meet the stated objective.

- If you're writing a function, assign it a value!
- Be sure module functions correctly. Test it thoroughly.

Make the module robust.

- Use language and system features which are provided to minimize errors:
 - Use `OPTIONS /EXTEND_SOURCE`.
 - Use `IMPLICIT NONE`.
 - Reference the disk area for `INCLUDE` files using a logical name.
 - Declare received character variables in subprograms as passed-length rather than fixed-length.
 - Prefer single quotes (') to Hollerith (nH) notation.
- Don't assume good data---do error checking---make module robust---reduce vulnerability---check arguments.
- Do not simply silently handle error conditions---report weirdness as discovered.
- Include variable (optional) run-time debugging output.

Documentation

Provide an overall description of the module in the header.

- First line should specify name of the module.
- State if the module is used as part of a system or is generic in nature.
- State the objective of the module.
- Explain input/output parameters (meaning, format, and limitations).
- Identify where implicit inputs and outputs reside (what `COMMON` area, etc.).
- Document the purpose/use of `INCLUDE` files.
- Document rules for validation.
- Define dependent codes and authorities.
- Describe the purpose of subprograms called.
- Document the purpose/use of logicals.
- Explain acronyms referenced.
- Note modifications so that modification history is clear. Specify the DAR number and a short code that will be used in the body to identify that modification (e.g., initials and modification number).
- Include instructions for module modification for correct maintenance.
- Update the header when maintaining the module.

Provide section comments in the body.

- Comment (blocks/sequences of) operations inside the module.
- Provide in-line comments often and where needed.
- Document complex operations especially carefully, with indentation as appropriate.
- When maintaining, prefer commenting out a block of code to deleting the block. If a block of code is commented out, give an explanation.
- Comment each modification using the identifying code given in the modification history portion of the header.

Readability/Understandability

Follow a consistent standard.

Formatting

- Indent to show dependent actions, e.g., inside IF - ELSE and DO WHILE. Don't indent just part of it.
- Try to keep one statement per line, at most.
- Use blank lines to separate blocks and logical parts.
- Indent using TAB unless there is lots of embedding. If lines get too short, indent with spaces so more will fit on a line.
- Don't cramp your intra-line style. Use blanks/white space to separate tokens. Be consistent in spacing.
- Clearly distinguish code from comments. One way to do this is to use all CAPS for code and mixed case for comments.
- When maintaining code, follow existing style. If code is all upper case, don't modify with lower case.

Naming

- Use logical names (rather than input/output unit numbers) for disk area locations on filenames.
- Use meaningful variable names. Use a name that suggests the variable's usage.
- Use meaningfully named "loop-controls" and "decision makers." Avoid X, I, J, etc.
- Do not use constant values for system status values, keystroke codes, or condition codes returned by system functions. Use the correct system names for these codes and INCLUDE the appropriate system library modules to define the names.
- Design user conventions (return codes, success/failure values, naming conventions, etc.) to be consistent with system conventions wherever possible.
- Use (user) parameter names and (user) variable names instead of constants or constant expressions if the constants have meaning.
- Group variables of the same type together in declarations. One variable per line and alphabetize.

Ease of Maintenance

Use modular design.

- Use system Run Time Library utility routines as much as possible. Don't reinvent.
- Reuse routines as much as possible.
- Module should only have one purpose.
- Actions invoked should be from a primary main area (i.e., a driver).

Use structured programming.

- Don't jump around.
- Use DO WHILE / END DO rather than GO TO and numbers.
- Program should have a single exit. Avoid multiple STOPS and RETURNS.

Use logic that is easy to follow.

- Avoid statement labels that are never used.
- If a multi-valued "CASE" statement (IF...ELSE IF..., etc.) is unavoidable, arrange the cases in a natural order, if such an order exists.
- When performing comparisons, use the logical operators that are most natural and efficient. Example: don't use 5 EQ's if 1 NE will suffice.

Design code to be flexible, for maximum usage with minimum modification.

- Make required input as general as possible---do not unnecessarily restrict choices for users.
- In setting sizes on variables to hold unknown input, don't be too skimpy---allow for very peculiar cases and then test input to ensure it matches your assumptions.

Efficiency

- Use IF-ELSE IF structure(s), rather than a series of IF THEN structures.
- Make repetitive operations involving data table-driven rather than hardcoded.
- Inside the code, reuse workspace variables wherever possible.
- Factor out repeated "IF" conditions to reduce code volume.
- Use statement functions to represent commonly used operations when appropriate.

SAMPLE HEADER

```

OPTIONS/EXTEND_SOURCE
!++
! PROGRAM NAME: CHECKINDEX.FOR
! APPLICATION SYSTEM/SUBSYSTEM: DUPCHK
! LANGUAGE: FORTRAN
! FUNCTIONAL DESCRIPTION:
!   CHECKINDEX checks all the records in an DUPCHK index file (either
!   AUTHOR or TITLE ) which match the key "KEYVAL" looking for a duplicate
!   of the current item.
! CALLING SEQUENCE:
!   CALL CHECKINDEX ( CHAN, KEYNO )
! FORMAL ARGUMENT(S):
! CHAN
!   VMS USAGE      longword, unsigned
!   TYPE           longword, unsigned
!   ACCESS         read only
!   MECHANISM      by reference
!
!   The i/o unit number (or channel number) of the DUPCHK database to
!   be checked.
! KEYNO
!   VMS USAGE      longword, unsigned
!   TYPE           longword, unsigned
!   ACCESS         read only
!   MECHANISM      by reference
!
!   The index to check.
!   "0" for the author index
!   "1" for the title index.
! IMPLICIT INPUT(s):
!   "CMP KEYS" - The compressed keys for the current
!   record. The compressed keys are stored in the
!   DUPCHK database and are used as the index.
!   CMP_KEYS is defined include file "DUPCHK.INC".
!
!   The "K ????" variables are the generated keys for the
!   "current item". Defined in the include file
!   "DUPKEYS.INC".
!
!   "MK ????" variables are generated keys for the "Master"
!   record expanded from the compressed keys in the DUPCHK
!   database. Defined in the include file "DUPKEYS.INC"
! IMPLICIT OUTPUT:
!   "ISADUP" - Actually set indirectly via a call to
!   COMPARE. If COMPARE determines that the current item
!   is a duplicate of a master item, then this routine
!   will return and "ISADUP" will be ".TRUE.".
!
!   "N AUTHOR RETRIEVALS" and "N TITLE RETRIEVALS" --
!   keeps track of the number of retrievals from the
!   author and title indexes for debugging and reporting
!   Defined in the include file "DUPKEYS.INC".
! SUBPROGRAMS CALLED:
!   COMPARE
!   DUMPKEYS
!   EXPAND
!   START_TIMER
!   STOP_TIMER
! CONDITION VALUES RETURNED: N/A
! DESCRIPTION:
!   This routine checks all the records in an index file (Author or Title)
!   which match the key "KEYVAL" until it finds a duplicate of the current
!   item or until there are no other records to match the key. KEYVAL is set
!   to the appropriate index key -- either AUTHOR // DOCUMENT TYPE or
!   TITLE // DOCUMENT TYPE depending on the value of KEYNO. The routine reads
!   through all of the records with this key, expands the compressed keys to
!   the expanded key form (MK ??? variables) and calls COMPARE to compare the
!   keys to the keys of the current item. If the record is determined to be a
!   duplicate of any record in the database, "ISADUP" is set .TRUE. and the
!   routine will return.
! ORIGINAL AUTHOR: [REDACTED]
!                   Science Applications International Corporation (SAIC)
! CREATION DATE: 1986
! EDIT HISTORY:
!
! Ver-Ed      Date      Who      What
!   001      11/88      [REDACTED]  Change K IDNUM to K IDNUM and
!                                   K IDYEAR to handle 11-digit
!                                   id number which is too large for
!                                   one longword integer.
! --

```

A variety of VAX FORTRAN facts, tips, suggestions and examples is maintained on the OSTI VAX Cluster in the file SAIC:FORTRAN.MISC.

Appendix G. KUB Software Quality Rule Set

"Software quality" in the KUB environment means development and maintenance of well-documented programs that are correct, readable, understandable, and efficient. The following guidelines provide general guidance to support software quality. They are not absolute; common sense should prevail.

Documentation

Documentation in general:

- New programs should have line numbers.

Documentation in Remarks Section:

- Remarks section should contain purpose of program, special notes about restrictions, run procedures, rerun procedures, update instructions, explanations of UPSI switches and anything else particular to the program.
- Admit to your work. (Use the author line.)

Documentation in Procedure Division:

- Comment paragraphs, logical sections, and complex lines of code.
- Enclose comments in boxes prior to the logic for obvious separation from code.

Documentation for Revisions:

- Remarks section should contain revision dates, reason for revision, and author's initials, and should follow the format of existing revision remarks.
- When lines of code are commented out, put in a comment box to explain why.
- Delete commented code when it becomes obsolete.
- Renumber after each major change.

Readability

Readability in general:

- Consistency of style is more important than use of any particular style. Be consistent with use of indentation, blank lines, comment style, blocking of statements, etc.

Readability of Procedure Division:

- Separate code into as many paragraphs as needed for readability and clarity.
- Use blank lines to separate sections of logic.
- Clearly define a break between the main body of the Procedure Division and any PERFORMed routines that are at the end of the program.
- Separate PERFORMed procedures from each other with a blank line.

Readability of Paragraphs:

- Indentation of code from paragraph names is important and the lines should line up. Indent a given number of positions. For continuation lines or nested statements, the next line should be further indented the same number of positions.

Readability of Statements:

- Code one statement per line.
- Clearly block individual statements. Don't make someone search for the end of the sentence.
- Use parenthesis where helpful on IF statements with multiple conditions.

Understandability

Understandability of Data:

- Paragraph names, file names, field names, UPSI names, and variable names are a means of documentation and should meaningfully describe what they represent.

Understandability of Flow of Control:

- Use a logical program structure. Put important actions in separate paragraphs. Examples: READ, MOVE, RETURN, DELETE, RELEASE, WRITE.
- Insert paragraph names at times when necessary to make easier to follow flow.
- Clearly block IF/ELSE statements to set apart the conditions.

Other Recommended Practices

- Use "AFTER ADVANCING" on printer WRITES.
- Since random reads are time consuming, use sequential access for extensive reading of VSAM files.
- On any VSAM I-O, make sure that the status code is checked. On an error condition, display the status code and record key.
- On any SQL command within a CICS program, move I-O description to the STEP-DENOTER working storage field for error handling.
- Avoid unnecessary repetition of variable names and operators in compound conditions.
- Put frequently-used statements (or groups of statements) in separate routines to be PERFORMed .
- A separate heading line for "KNOXVILLE UTILITIES BOARD" is necessary only in rare instances. Do not install it unless directed.
- On all stock paper printouts, the program name should be printed at the extreme right end of the first heading line. Exception: When the user will not accept it.
- On all programs which will update a disk file, install an UPSI to avoid an update.
- When using the COPY statement for long FD or 01 entries, use the COPY SUPPRESS option.
- Do not write your own date routines. The NXDATE routine will handle all date situations.
- When cataloging a new FD entry, always include the RECORD CONTAINS clause.
- Unless necessary to do otherwise, use only one OPEN and CLOSE per program.
- For all programs with an UPSI, code a display statement which will cause a single line to display on the log showing all options in effect during that execution of the program. Do not display if none in effect.
- On all even-numbered numeric fields to be packed, increase to next odd. It costs no more space and is more efficient.

Additional Guidelines to be added by KUB ISD staff

Further Information

See the "Programming Techniques" section of the IBM DOS/VS COBOL Compiler and Library Programmer's Guide.

VITA

Carmen J. Trammell was born in Oak Ridge, Tennessee, on June 29, 1952. She graduated from Oak Ridge High School in June of 1970, and from Vanderbilt University with a B.A. in Philosophy in December of 1973.

In 1974, Ms. Trammell began work for Union Carbide Corporation's Nuclear Division in Oak Ridge, Tennessee. She held a variety of human resource management positions during her tenure with Union Carbide from 1974 to 1981. During that time, she pursued graduate studies in clinical and industrial/organizational psychology at the University of Tennessee, and received the Doctor of Philosophy in Psychology degree in August of 1981.

During the early 1980s, Ms. Trammell continued her work in human resource management with Atari, Inc., in Sunnyvale, California, and Hewlett-Packard Company in Palo Alto, California. On returning to Tennessee in 1985, she worked as a management consultant for Executive Learning, Inc., and taught in an undergraduate program in Business Administration for Tusculum College.

Ms. Trammell entered the graduate program in Computer Science at the University of Tennessee in September of 1987. While pursuing an M.S. degree in Computer Science with a specialization in software engineering, she worked in software development for the University of Tennessee, Martin Marietta Energy Systems, and Software Engineering Technology, Inc. She was awarded the Master of Science degree in Computer Science in May of 1991.