

Technical Report TR-688  
SEL-2

August 1978  
AFOSR-77-3181A

INVESTIGATING SOFTWARE  
DEVELOPMENT APPROACHES \*

Victor R. Basili and Robert W. Reiter, Jr.

Department of Computer Science  
University of Maryland  
College Park, MD 20742

---

\* Research supported in part by the Air Force Office of Scientific Research grant AFOSR-77-3181A to the University of Maryland. Computer time supported in part through the facilities of the Computer Science Center of the University of Maryland. The material contained in this paper will become part of a dissertation to be submitted to the Graduate School, University of Maryland, by Robert W. Reiter, Jr., in partial fulfillment of the requirements for the Ph.D. degree in Computer Science.

© Copyright 1978 by V. R. Basili and R. W. Reiter, Jr.

**Abstract:**

This paper reports on research comparing various approaches, or methodologies, for software development. The study focuses on the quantitative analysis of the application of certain methodologies in an experimental environment, in order to further understand their effects and better demonstrate their advantages in a controlled environment. A series of statistical experiments were conducted comparing programming teams that used a disciplined methodology (consisting of top-down design, process design language usage, structured programming, code reading, and chief programmer team organization) with programming teams and individual programmers that employed ad hoc approaches. Specific details of the experimental setting, the investigative approach (used to plan, execute, and analyze the experiments), and some of the results of the experiments are discussed.

**Key Words and Phrases:**

analysis of software development, disciplined methodology, programming teams, experimental study, program measurements, software metrics

CR Categories: 4.6

Table of Contents

Acknowledgements . . . . .	v
Section	
I. Introduction . . . . .	1
II. Specifics . . . . .	6
Design/Setup . . . . .	6
Environment . . . . .	10
Data Collection and Reduction . . . . .	12
Programming Aspects and Metrics . . . . .	13
III. Approach . . . . .	20
Step 1: Questions of Interest . . . . .	21
Step 2: Research Hypotheses . . . . .	23
Step 3: Statistical Model . . . . .	24
Step 4: Statistical Hypotheses . . . . .	25
Step 5: Research Frameworks . . . . .	26
Step 6: Experimental Design . . . . .	30
Step 7: Collected Data . . . . .	30
Step 8: Statistical Test Procedures . . . . .	30
Step 9: Statistical Results . . . . .	32
Step 10: Statistical Conclusions . . . . .	34
Step 11: Research Interpretations . . . . .	37
IV. Objective Results . . . . .	39
Presentation . . . . .	39
Impact Evaluation . . . . .	40
A Relaxed Differentiation View . . . . .	48
A Directionless View . . . . .	49
Individual Highlights . . . . .	51
V. Interpretive Results . . . . .	55
According to Basic Suppositions . . . . .	55
According to Programming Aspect Classification . . . . .	60
VI. Concluding Remarks . . . . .	79
References . . . . .	81
Appendix	
1. Explanatory Notes for the Programming Aspects . . . . .	84
2. English Statements for the Non-Null Conclusions . . . . .	98
3. English Paraphrase of Relaxed Differentiation Analysis . . . . .	106
4. English Categorization of Directionless Distinctions . . . . .	107

## List of Diagrams and Tables

## Diagram

1.	Approach Schematic . . . . .	22
2.1	Lattice of Possible Directional Outcomes for Three-way Comparison . . . . .	28
2.2	Lattice of Possible Nondirectional Outcomes for Three-way Comparison . . . . .	28
3.	Association Chart for Results and Conclusions . . . . .	36

## Table

1.	Programming Aspects . . . . .	15
2.1	Non-Null Conclusions, for Location Comparisons, arranged by outcome . . . . .	41
2.2	Non-Null Conclusions, for Dispersion Comparisons, arranged by outcome . . . . .	42
3.	Statistical Conclusions . . . . .	43
4.1	Relaxed Differentiation for Location Comparisons . . . . .	50
4.2	Relaxed Differentiation for Dispersion Comparisons . . . . .	50
5.1	Conclusions for Class I, Effort (Job Steps) . . . . .	61
5.2	Conclusions for Class II, Errors (Program Changes) . . . . .	61
5.3	Conclusions for Class III, Gross Size . . . . .	62
5.4	Conclusions for Class IV, Control-Construct Structure . . . . .	63
5.5	Conclusions for Class V, Data Variable Organization . . . . .	64
5.6	Conclusions for Class VI, Packaging Structure . . . . .	65
5.7	Conclusions for Class VII, Invocation Organization . . . . .	65
5.8	Conclusions for Class VIII, Communication via Parameters . . . . .	66
5.9	Conclusions for Class IX, Communication via Global Variables . . . . .	67

### Acknowledgements

It is a pleasure to acknowledge colleagues Dr. John D. Gannon (University of Maryland) and Dr. Herbert E. Dunsmore (Purdue University) for their helpful comments and stimulating discussions. The authors are indebted to Dr. Richard H. Meltzer (General Electric) and Mr. W. Douglas Brooks (IBM Federal Systems) for their assistance in keeping the statistical side of the investigation fair and square. Mrs. Claire Bacigaluppi is appreciated for having expertly typed numerous drafts of this report.

## I. Introduction

In the development of any theory, there are three phases of validation. First is the logical development of the theory based on a set of sound principles. Second is the application of the theory and the gathering of evidence that the theory is applicable in practice. This usually involves some qualitative assessment in the form of case studies. The third step is the quantitative analysis of the application of theory in an experimental environment in order to further understand its effect and better demonstrate its advantages in a controlled environment.

There has been a great deal written about methodologies and programming environments for developing software [Wirth 71; Dahl, Dijkstra, and Hoare 72; Jackson 75; Brooks 75; Myers 75; Linger, Mills, and Witt 79]. It is clear that many of these methodologies are based on sound logical principles and their adoption within a production environment has been successful. There have been many case studies that attempt to validate these theories; projects have adapted versions of these methods and have reported varying degrees of success, i.e., the users feel they got the job done faster, made less errors, or produced a better product [Baker 75; Basili and Turner 75; Daley 77]. Unfortunately, there has been a minimum of real quantitative evidence that comparatively assesses any particular methodology [Shneiderman et al. 77; Myers 78; Sheppard et al. 78]. This is partially because of the cost and impracticality of a valid experimental setup within a production ("real-world") environment.

This leaves open the question of whether there is a measurable benefit derived from various programming methodologies and environments with respect to either the developed product or the development process. Even if the benefits are real, it is not clear that they can be quantified and effectively monitored. Software development is still too much of an art in the aesthetic or spontaneous sense. In order to fully understand it, control

it, and adapt it to particular applications and situations, software development must become more of a science in the engineering and calculated sense. What is required is more empirical study, data collection, and experimental analysis.

The purpose of the research reported in this paper is (a) to quantitatively investigate the effect of methodologies and programming environments on software development and (b) to develop an investigative methodology based on scientific experimentation and tailored to this particular application. It involves the measurement and analysis of both the process and the product in a manner which is minimally obtrusive (to those developing the software), very objective, and highly automatable. The goal of the research was to verify the effectiveness of a particular programming methodology and to identify various quantifiable aspects that could demonstrate such effectiveness.

To this end, a controlled experiment was conducted involving several replications of a specific software development task under varying programming environments. For each replication successive versions of the software under development were entered in an historical data base which recorded details of the development process and product. A host of measurements were extracted from the data base and statistically analyzed in order to achieve the research goals. Some of these measurements were "confirmatory", as they were planned in advance and expected to show differences among the programming environments being investigated, while many of the measurements were simply "exploratory."

The study involves three distinct groupings of software developers: individual programmers, ad hoc three-person programming teams, and three-person programming teams using a disciplined methodology. The individual programmers and the ad hoc teams were allowed to develop the software in a manner of their own choosing; this is referred to as an ad hoc approach. The disciplined methodology referred to in this paper consists of an integrated set of software development techniques and team

organizations which include top-down design, use of a process design language, structured programming, code reading, and chief programmer teams.

The study examines differences in the expectency and the predictability of software development behavior under the programming environments represented by these groups.

The basic premise is that distinctions among the groups exist both in the process and in the product. With respect to the software development product, it is believed that the disciplined team should approximate the single individual with regard to product characteristics (such as number of decisions coded and global data accessibility) and at the very least lie somewhere between the single individual and the ad hoc team. This is because the disciplined methodology should help in making the team act as a mentally cohesive unit during the design, coding, and testing phases. With respect to the software development process, the disciplined team should have advantages over both individuals and ad hoc teams, displaying superior performance on cost factors such as computer usage and number of errors made. This is because of the discipline itself and because of the ability to use team members as a resource for validation.

The study's findings reveal several programming characteristics for which statistically significant differences do exist among the groups. The disciplined teams used fewer computer runs and apparently made fewer errors during software development than either the individual programmers or the ad hoc teams. The individual programmers and the disciplined teams both produced software with essentially the same number of decision statements, but software produced by the ad hoc teams contained greater numbers of decision statements. For no characteristic was it concluded that the disciplined methodology impaired the effectiveness of a programming team or diminished the quality of the software product.



The investigation has been conducted in a laboratory or proving-ground fashion, in order to achieve some statistical significance and scientific respectability without sacrificing production realism and professional applicability. By scaling down a typical production environment while retaining its important characteristics, the laboratory setting provides for a reasonable compromise between the extremes of

(a) "toy" experiments,

which can afford elaborate experimental designs and large sample sizes but often suffer from a basic task that is rather unrelated to production situations or involve a context from which it is difficult to extrapolate or scale up (e.g., introductory computer course students taking multiple-choice quizzes based on thirty-line programs),

and (b) "production environment" experiments,

which offer a high degree of production realism by definition but incur prohibitively high costs even for the simplest and weakest experimental designs (i.e., statistical experimentation requires replication, and multiple duplication of a nontrivial programming project is clearly expensive).

The experiment in this study was conducted within an academic environment where it was possible to achieve an adequate experimental design and still simulate key elements of a production environment.

An initial phase of investigation has been completed and the complete results are presented in the remainder of this paper. Section II gives details pertaining to the experiment itself. Section III describes the investigative methodology used to plan, execute, and analyze the experiment. Sections IV and V present the experiment's results, segregated into empirical findings (resulting from statistical analysis of the measurements) and intuitive judgements (resulting from interpretation of some of the empirical findings), respectively. Section VI contains some remarks on this initial phase of investigative effort and a discussion of further work planned for the study.

It should be noted that the terms 'methodology' and 'methodological' (in reference to software development) are consistently used throughout this report with a technical meaning related to the concept of a comprehensive integrated set of development techniques as well as team organizations, rather than to the more common notion of a particular technique or organization in isolation.

## II. Specifics

This section describes several aspects of the experiment itself; namely, the experimental design or setup, the experimental environment, data collection and reduction (during and subsequent to the experiment), and the programming aspects and associated metrics (used to quantify the experiment).

### Design/Setup

The major facets of the experimental design are the experimental units, the experimental treatment factors, the experimental treatment factor levels, the experimental variables observed, the experimental local control, and the experimental management of other factors. (See [Ostle and Mensing 75; Chapter 9] for a thorough presentation of these facets.) An experimental unit is that unit to which a single treatment (which may be a combination of several factor levels) is applied in one replication of the basic experiment. In this case, the basic experiment was the accomplishment of a given software development project, and the experimental unit was the software development team, i.e., a small group of people who worked together to develop the software. There was a total of 19 such units involved in the experiment.

In most experiments, attention is focused on one or more independent variables and on the behavior of a certain dependent variable(s) as the independent variables are permitted to vary. These independent variables are known as experimental treatment factors. This experiment focused on two particular facets of software development, (1) size of the development team and (2) degree of methodological discipline, as the experimental treatment factors.

Most experiments involve some deliberate differential variation in the experimental treatment factors. The various

values or classifications of the factors are known as the levels of the experimental treatment factors. In this experiment, two levels were selected for each factor. For the size factor, the levels were single individuals and three-person teams. For the degree-of-discipline factor, the levels were an ad hoc approach and a disciplined methodology.

The experimental (dependent) variables observed consisted of 130 programming aspects relating to the software product and development process. Technically, this created a whole series of simultaneous univariate experiments, all having the same common experimental design and all based on the same data sample, with one experiment for each programming aspect. The immediate goal of an experiment is to learn something about the relationship between the experimental treatment factor levels and the observed variables.

Experimental local control refers to the configuration by which (a) experimental units are obtained, (b) certain sets of units are placed into groups, and (c) these different groups are subjected to certain combinations of experimental factor levels. Local control is employed in the design of an experiment in order to increase the statistical efficiency of the experiment (or sensitivity/power of the statistical test). Experimental local control usually incorporates some form(s) of randomization -- a basic principle of experimental design -- since it is necessary for the validity of statistical test procedures.

For this experiment, subjects were obtained simply on the basis of course enrollment. Since the experiment was completely embedded within two academic courses, every student in those courses automatically participated in the experiment. Development "teams" were formed among the subjects: in one course, the students were allowed to choose between segregating themselves as individual programmers or combining with two other classmates as three-person programming teams; in the other course, the students were assigned (by the researchers) into three-person teams.

Experimental units were formed and placed into groups in this manner because the two academic courses themselves provided the two levels of the second experimental treatment factor. This process yielded three groups of 6, 6, and 7 units, designated AI, AT, and DT, respectively. Each group was exposed to a particular combined factor-level treatment according to the following partial factorial arrangement: (AI) single individuals using an ad hoc approach, (AT) three-person teams using an ad hoc approach, and (DT) three-person teams using specific state-of-the-art methodologies.

The disciplined methodology imposed on teams in group DT consisted of an integrated set of techniques, including top down design of the problem solution using a Process Design Language (PDL), functional expansion, design and code reading, walk-throughs, and chief programmer and manager teams. These techniques and organizations were taught as an integral part of the course that the subjects were taking. The course material was organized around [Linger, Mills, and Witt 79], [Basili and Baker 75], and [Brooks 75] as textbooks. Since the subjects were novices in the methodology, they executed the techniques and organizations to varying degrees of thoroughness and were not always as successful as seasoned users of the methodology would be.

Specifically, the disciplined methodology prescribed the use of a PDL for expressing the design of the problem solution. The design was expressed in a top-down manner, each level representing a solution to the problem at a particular level of abstraction and specifying the functions to be expanded at the next level. The PDL consisted of a specific set of structured control and data structures, plus an open-ended designer-defined set of operators and operands corresponding to the level of the solution and the particular application. Design and code reading involved the critical review of each team member's PDL or code by at least one other member of the team. Walk-throughs represented a more formalized presentation of an individual's work to the other

members of the team in which the PDL or code was explained step by step. In the chief programmer teams, the chief programmer defined the top level solution to the problem in PDL, designed and implemented the key code himself, and assigned subtasks to each of the other two programmers who code read for the chief programmer, designed or coded subpieces as requested by him, and performed librarian activities (i.e., entering or revising code stored on-line, making test runs, etc.). The manager teams were defined in a similar fashion, except that the manager also acted as librarian, writing less code and doing more code reading, and yielded much greater responsibility for design and implementation to the other members of the team.

Each individual or team in groups AI and AT was allowed to develop the software in a manner of their own choosing, which is referred to in this paper as an ad hoc approach. No methodology was taught in the course these subjects were taking. Informal observation by the experimenters confirmed that the approaches used by the individuals and ad hoc teams were indeed lacking in discipline and did not utilize the key elements of the disciplined methodology (e.g., an individual working alone cannot practice code reading, and it was evident that the ad hoc teams did not use a PDL or formally do a top-down design).

There are usually several extraneous factors, other than the ones identified as experimental treatment factors, which could influence the behavior being observed. The experimental design employed three distinct methods to control various extraneous factors. Factors were either fixed (artificially or externally held constant across all experimental units), balanced (artificially or externally distributed as evenly as possible among the experimental units), or randomized (allowed to vary in a naturally random way among the experimental units). In this experiment, a variety of programming factors which do affect software development were given conscious consideration as extraneous variables and controlled as follows:

- personal ability/talent of people: randomized

(and balanced within disciplined teams)

- project/task/application: fixed
- project specifications: fixed
- implementation language: fixed
- calendar schedule: fixed
- available computer resources: fixed
- available man-hour resources: randomized
- available automated tools: fixed

Wherever possible, these variables were held constant by explicitly treating all experimental units in the same manner. Two variables, the personal ability of the participants and the amount of actual time they (as students with other classes and responsibilities) had to devote to the project, could only be allowed to vary among the groups in what was assumed to be a random manner. However, information from a questionnaire was used to balance the personal ability of the participants in the disciplined teams (only) by first (a) partitioning the group DT students into three equal-sized categories (high, medium, low) based on their grades in previous computer courses and their extracurricular programming experience, and then (b) assigning them to teams by randomly selecting one student from each category to form each team.

### Environment

Several particulars of the experimental environment contribute significantly to the context in which the experiment's results must be appraised. These include the time and place the experiment was conducted, the software development project (or application) which served as the task performed during the experiment, the people who participated as subjects, and the computer programming language in which the software was written.

The experiment was conducted during the Spring 1976 semester, January through May, within the regular academic courses given by the Department of Computer Science on the College Park campus of the University of Maryland.

Several general characteristics of the project are noteworthy. The application was a compiler, involving string processing and language translation (via scanning, parsing, code generation, and symbol table management). The scope of the project excluded both extensive error handling and user documentation. The project difficulty was slight but nonnegligible, requiring roughly a two man-month effort. The size of the resulting system averaged over 1200 lines of high-level (structured language) source code. The total task was to design, implement, test, and debug the complete computer software system given a particular specification. All aspects of the project were fixed and uniform for each of the development teams. Each team worked independently to build its own system, using identical (1) specifications, (2) computer resources allocated, (3) duration of calendar time allotted, (4) implementation language, (5) testing tools, etc.

The participants were advanced undergraduate students and graduate students in the Department of Computer Science. None were novice programmers, all had completed at least four semesters of programming course work, several were about to graduate and take programming jobs in government or industry, and a few even had as much as three years' professional programming experience. On the whole, the participants might best be described as "advanced student programmers with a bit of professional experience." The experiment was conducted within the framework of two comparable advanced elective courses, each with the same academic prerequisites. The project and the experimental treatments were built into the course material and assignments, and everyone in the two classes participated in the experiment. They were aware of being monitored, but had no knowledge of what was being observed or why. A reasonable degree of homogeneity seemed to exist among the participants with respect to personnel factors, such as ability, experience, motivation, time/effort devoted to the project, etc. On the whole, they were typically average in each of these factors with natural fluctuations which appeared to be evenly distributed among the experimental groups in



a random fashion. Based upon pre-experiment qualitative judgment, all subjects shared a similar background with respect to team programming and the disciplined methodology. However, groups AI and AT (the individuals and ad hoc teams) seemed to have had a slight edge over group DT (the disciplined teams) with respect to general programming ability and formal training in the application area.

The implementation language was the high-level, non-block-structured, structured-programming language SIMPL-T [Basili and Turner 76]. This language was designed and developed at the University of Maryland where it is taught and used extensively in regular Department of Computer Science courses. It is characterized by a very simple and efficient run-time environment. SIMPL-T contains the following control constructs: sequence, ifthenelse, whiledo, case, and exits from loops (but no gotos). The language adheres to a philosophy of "strong data typing" and all variables must be explicitly declared. It provides the programmer with both automatic recursion and string-processing capabilities similar to PL/1.

#### Data Collection and Reduction

Due to the partially exploratory nature of the experiment in terms of differences to be discovered in the project and process, as much information was collected as could be done in an efficient, effective, and unobtrusive manner. A variety of information sources was used. Individual questionnaires supplied the personal background and programming experience of each participant. Private team interviews and open-class team reports yielded information regarding individual performance on the project. Run logs and computer account billing reports gave a record of the computer activity during the project. Special module compilation and program execution processors (invoked on-line via very slight changes to the regular command language) created an historical data base of source code and test data accumulated throughout the project development.

The data base provided the principal source of information analyzed in the current investigation and other information sources have been utilized only in an auxiliary manner (if at all). Thus, data collection for the experiments themselves was automated on-line, with essentially no interference to the programmer's normal pattern of actions during computer (terminal) sessions. The final products were isolated from the data base and measured for various syntactic and organizational aspects of the finished product source code. Effort and cost data were also extracted from the data base. The inputs to the analysis, in the form of scores for the various programming aspects, reflect the quantitatively measured character of the product and effort of the process. Much of the data reduction was done automatically within a specially instrumented compiler. Some was done manually (e.g., examining characteristics across modules). Due to the underlying collection and reduction mechanism, which was uniformly applied to all experimental units, the data used in the analysis has the characteristics of objectivity, uniformity, and quantitiveness and is measured on an interval scale of measurement [Conover 71; pp. 65-67].

### Programming Aspects and Metrics

The dependent variables studied in this experiment are called programming aspects. They represent specific isolatable and observable features of the programming phenomenon which are highly automatable (i.e., they could be extracted or computed directly on-line from information readily obtainable from operating systems and compilers). The variables fall into two categories: process aspects and product aspects. Process aspects are related to characteristics of the development process itself, in particular, the cost and required effort as reflected in the number of computer job steps (or runs) and the amount of textual revision of source code during development. Product aspects are related to the syntactic content and organization of the symbolic source code which represents the complete final product that was developed. Examples are number of lines, frequency of particular statement

types, average size of data variables' scope, etc. For each aspect there exists an associated metric, a specific algorithm which ultimately defines that aspect and by which it is measured.

The particular programming aspects examined in this investigation are listed in Table 1. They appear grouped by category; indented qualifying phrases specify particular variants of certain general aspects. When referring in this paper to an individual (sub)aspect, a concatenation of the heading line with the qualifying phrases (separated by \ symbols) is used; for example, COMPUTER JOB STEPS\MODULE COMPILATIONS\UNIQUE denotes the number of COMPUTER JOB STEPS that were MODULE COMPILATIONS in which the source code was UNIQUE from all other compiled versions. Explanatory notes (keyed to the list in Table 1) about the programming aspects are given in Appendix 1, complete with definitions for the nontrivial or unfamiliar metrics. Technical meanings for various system- or language-dependent terms used in the paper (e.g., module, segment, intrinsic, entry) also appear there. Some of these words mean different things to different people, and the reader is cautioned against drawing inferences not based on this paper's definitions.

The complete set of programming aspects may be partitioned into two subsets based upon the motivation for their inclusion in the study. Several aspects --hereafter denoted as "confirmatory"-- had been consciously planned in advance of collecting and extracting the data, because intuition suggested that they would serve well as quantitative indicators of important qualitative characteristics of the software development phenomenon. It was predicted a priori that these "confirmatory" aspects would verify the study's basic premises regarding the programming methodologies being investigated in the experiment. The remaining aspects --hereafter denoted as "exploratory"-- were considered mainly because they could be collected and extracted cheaply (even as a natural by-product sometimes) along with the "confirmatory" aspects. There was little serious expectation that these "exploratory" aspects would be useful indicators of differences

Table 1. Programming Aspects

N.B. The asterisks to the left mark "confirmatory" aspects; "exploratory" aspects are unmarked. The parenthesized numbers to the right refer to the explanatory notes in Appendix 1.

*****	
	development process aspects :
	=====
*	COMPUTER JOB STEPS (1)
*	MODULE COMPILATIONS (2)
*	UNIQUE (3)
	IDENTICAL (3)
*	PROGRAM EXECUTIONS (4)
	MISCELLANEOUS (5)
	-----
*	ESSENTIAL JOB STEPS (6)
	AVERAGE UNIQUE COMPILATIONS PER MODULE (7)
	MAX UNIQUE COMPILATIONS F.A.O. MODULE (8)
	=====
*	PROGRAM CHANGES (9)
	*****
	final product aspects :
	=====
*	MODULES (10)
	=====
*	SEGMENTS (11)
	-----
	SEGMENT TYPE COUNTS :
	FUNCTION (11)
	PROCEDURE (11)
	-----
	SEGMENT TYPE PERCENTAGES :
	FUNCTION (11)
	PROCEDURE (11)
	-----
	AVERAGE SEGMENTS PER MODULE (13)
	=====
*	LINES (14)
	=====
*	STATEMENTS (15)
	-----
	STATEMENT TYPE COUNTS :
	:= (16)
	IF (17)
*	CASE (18)
*	WHILE (19)
*	EXIT (20)
*	EXIT (21)
	(PROC)CALL (22) (44)
	NONINTRINSIC (23) (44)
	INTRINSIC (23) (44)
*	RETURN (24)
	-----
	STATEMENT TYPE PERCENTAGES :
	:= (16)
*	IF (17)
*	CASE (18)
*	WHILE (19)
*	EXIT (20)
*	EXIT (21)
	(PROC)CALL (22)
	NONINTRINSIC (23)
	INTRINSIC (23)
*	RETURN (24)
	-----
*	AVERAGE STATEMENTS PER SEGMENT (25)
	=====
*	AVERAGE STATEMENT NESTING LEVEL (26)
	=====
*	DECISIONS (27)
	=====
	FUNCTION CALLS (22) (44)
	NONINTRINSIC (23) (44)
	INTRINSIC (23) (44)
	=====

* TOKENS	(28)
* AVERAGE TOKENS PER STATEMENT	(28)
=====	
INVOCATIONS	(29)
FUNCTION	(11) (44)
NONINTRINSIC	(23) (44)
INTRINSIC	(23) (44)
PROCEDURE	(11) (44)
NONINTRINSIC	(23) (44)
INTRINSIC	(23) (44)
NONINTRINSIC	(23)
INTRINSIC	(23)
-----	
AVG INVOCATIONS PER (CALLING) SEGMENT	(30)
FUNCTION	(11)
NONINTRINSIC	(23)
INTRINSIC	(23)
PROCEDURE	(11)
NONINTRINSIC	(23)
INTRINSIC	(23)
NONINTRINSIC	(23) (44)
INTRINSIC	(23)
-----	
AVG INVOCATIONS PER (CALLED) SEGMENT	(31) (44)
FUNCTION	(11)
PROCEDURE	(11)
=====	
DATA VARIABLES	(32)
-----	
DATA VARIABLE SCOPE COUNTS :	(37)
* GLOBAL	(33)
ENTRY	(34)
MODIFIED	(35)
UNMODIFIED	(35)
NONENTRY	(34)
MODIFIED	(35)
UNMODIFIED	(35)
MODIFIED	(35)
UNMODIFIED	(35)
* NONGLOBAL	(33)
PARAMETER	(33)
VALUE	(36)
REFERENCE	(36)
* LOCAL	(33)
-----	
DATA VARIABLE SCOPE PERCENTAGES :	(37)
* GLOBAL	(33)
ENTRY	(34)
MODIFIED	(35)
UNMODIFIED	(35)
NONENTRY	(34)
MODIFIED	(35)
UNMODIFIED	(35)
MODIFIED	(35)
UNMODIFIED	(35)
* NONGLOBAL	(33)
PARAMETER	(33)
VALUE	(36)
REFERENCE	(36)
* LOCAL	(33)
-----	
AVERAGE GLOBAL VARIABLES PER MODULE	(38)
ENTRY	(34)
NONENTRY	(34)
MODIFIED	(35)
UNMODIFIED	(35)
-----	
AVERAGE NONGLOBAL VARIABLES PER SEGMENT	(38)
PARAMETER	(33)
LOCAL	(33)
=====	

=====		
PARAMETER PASSAGE TYPE PERCENTAGES :		(39)
VALUE		(36)
REFERENCE		(36)
=====		
(SEG, GLOBAL) ACTUAL USAGE PAIRS		(40)
ENTRY		(34)
MODIFIED		(35)
UNMODIFIED		(35)
NONENTRY		(34)
MODIFIED		(35)
UNMODIFIED		(35)
MODIFIED		(35)
UNMODIFIED		(35)
-----		
(SEG, GLOBAL) POSSIBLE USAGE PAIRS		(40)
ENTRY		(34)
MODIFIED		(35)
UNMODIFIED		(35)
NONENTRY		(34)
MODIFIED		(35)
UNMODIFIED		(35)
MODIFIED		(35)
UNMODIFIED		(35)
-----		
* (SEG, GLOBAL) USAGE RELATIVE PERCENTAGES		(40)
ENTRY		(34)
MODIFIED		(35)
UNMODIFIED		(35)
NONENTRY		(34)
MODIFIED		(35)
UNMODIFIED		(35)
MODIFIED		(35)
UNMODIFIED		(35)
=====		
* (SEG, GLOBAL, SEG) DATA BINDINGS :		(41)
ACTUAL		(42)
SUBFUNCTIONAL		(43)
INDEPENDENT		(43)
* POSSIBLE		(42)
* RELATIVE PERCENTAGE		(42)
*****		

among the groups; but they were included in the study with the intent of observing as many aspects as possible on the off chance of discovering any unexpected tendency or difference. The "confirmatory" programming aspects are identified by being flagged in Table 1 with an asterisk; the "exploratory" programming aspects are unflagged.

This distinction between "confirmatory" and "exploratory" has important consequences for the evaluation of the study's experiments. For the "confirmatory" aspects, the individual experiments are actually confirmatory, since it was hypothesized that they would indicate certain differences among the groups, prior to conducting the experiment and extracting their scores. But for the "exploratory" aspects, whose scores were extracted without any preconceived hypotheses, the experiments are purely exploratory. Thus, this study combines elements of both confirmatory and exploratory data analysis within one common experimental setting [Tukey 69]. This distinction does not however influence the method by which the experiments themselves were conducted.

It should be noted that a large percentage of the product aspects fall into the "exploratory" category. A secondary motivation for their consideration is that the product aspects, as a unit, represent a fairly extensive taxonomy of the surface features of software. The idea that important software qualities (e.g., "complexity") could be measured by counting such surface features has generally been disregarded by researchers as too simplistic (e.g., [Mills 73; p. 232]). A resolve to study these surface features empirically, to see if something might turn up, before rejecting the underlying idea, was partially responsible for their inclusion in the study.

In order to avoid any inadvertent deception or misunderstanding, the following issue of redundancy must be stated and properly appreciated. There exist several instances of duplicate programming aspects; that is, certain logically unique

aspects appear a second time with another name, in order to provide alternative views of the same metric and to achieve a certain degree of completeness within a set of related aspects. For example, the FUNCTION CALLS aspect and the STATEMENT TYPE COUNTS\ (PROC)CALL aspect are listed (and categorized appropriately) from the viewpoint of the various type of constructs that comprise the implementation language. But the very same metrics can be considered from the unifying viewpoint of the various subtype frequencies for segment invocations, and thus it is desirable to include the duplicate aspects INVOCATIONS\FUNCTIONS and INVOCATIONS\PROCEDURES as part of the natural categorization of INVOCATIONS. Within the 137 programming aspects listed in Table 1, there are seven pairs of duplicate aspects (identified in the notes of Appendix 1), leaving 130 nonredundant aspects examined in the study. By definition, the data scores obtained for any pair of duplicate aspects will be identical, and thus the same statistical conclusions will be reached for both aspects. This must be kept in mind when evaluating the results of the experiments in terms of their statistical impact.



### III. Approach

This section describes the steps in an investigative methodology developed for the particular problem of comparing software development efforts under various conditions. It was used to guide the planning, execution, and analysis of the experimental investigations whose results are reported in this paper.

The investigative methodology can be characterized as an empirical study based on the "construction" paradigm in which multiple subjects are closely monitored during actual "production" experiences, each subject performing the same task, with controlled variation in specific variables. It uses scientific experimentation and statistical analysis based on a "differentiation among groups by aspects" paradigm in which possible differences among the groups, as indicated by differences in certain quantitatively measured aspects of the observed phenomenon, are the target of the analysis. This use of "difference discrimination" as the analytical technique dictates a model of homogeneity hypothesis testing that influences nearly every element of the methodology.

Note that there are other analysis techniques that could have been used; e.g., estimation of magnitude of difference, correlations between various aspects (across all combinations of factor-levels), multivariate analysis (rather than multiple univariate analyses in parallel), and factor analysis (breakdown of variance) among the various aspects. These are useful techniques and may be used in later phases of this research. However, difference discrimination represents a "first-cut" probe, which hopefully will yield some information to help guide more refined probes in the future.

Although the methodology is built around an empirical study and utilizes scientific experimentation, the actual execution of

the experiments and collection of data play a small role in the overall methodology when compared to the planning and analysis phases. This is readily apparent from the Approach Schematic, Diagram 1, which charts some of the relationships among the various elements (or steps) of the investigative methodology.

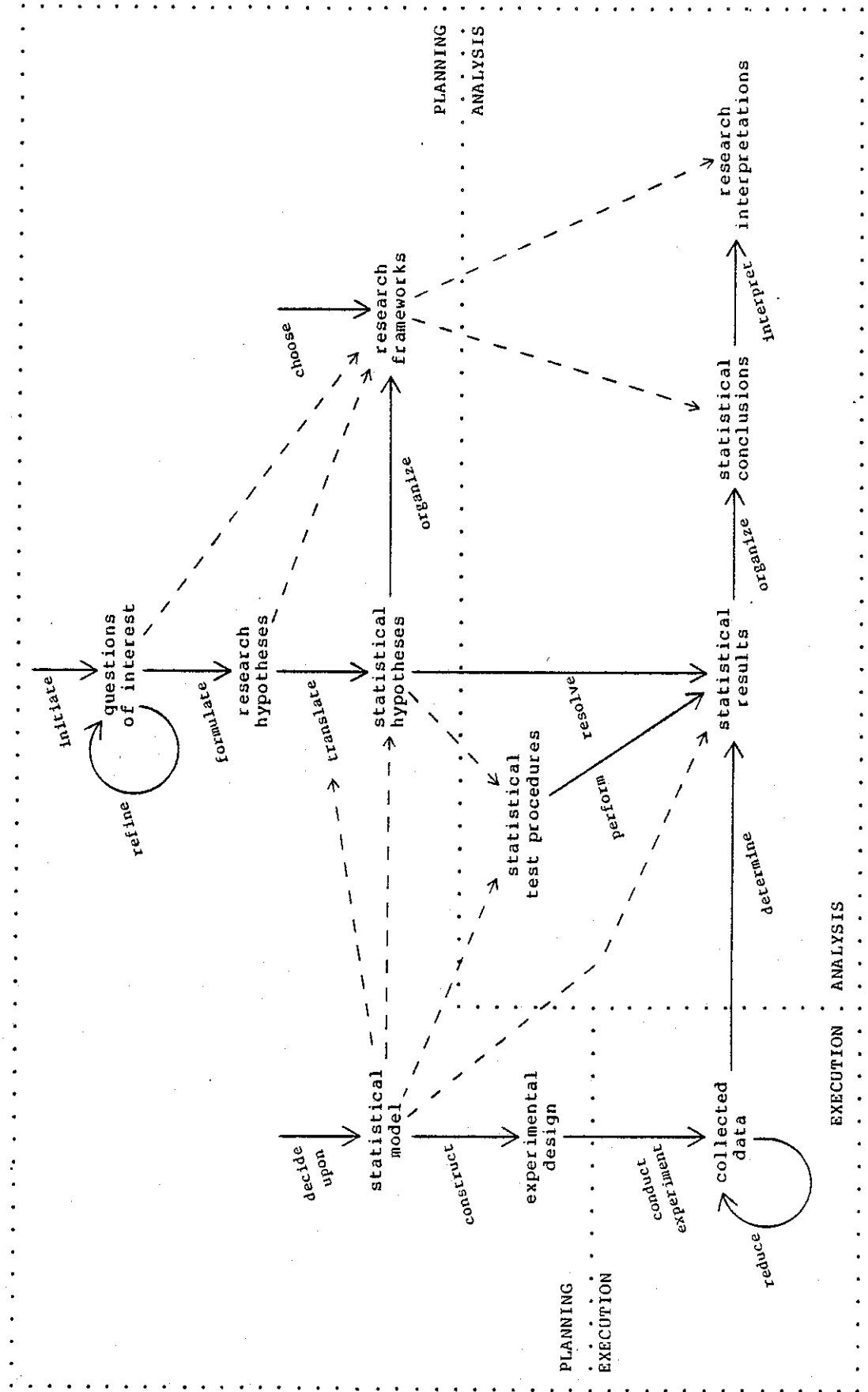
The remainder of this chapter outlines and briefly describes the overall approach by defining each step in general and discussing how the approach was applied in the research effort at hand. Note that Sections IV and V give the specifics of the last two steps of the approach -- statistical conclusions and research interpretations-- as pertaining to the current experimental investigation.

#### Step 1: Questions of Interest

Several questions of interest were initiated and refined so that answers could be given in the form of statistical conclusions and research interpretations. Questions were formulated on the basis of several areas of interest: (1) software development rather than software maintenance, (2) a particular set of programming factors, (3) quantitatively measurable aspects of the process and the product, (4) two particular levels for each of the programming factors, (5) the particular type of analysis technique mentioned above, and (6) intuitive considerations and suspicions leading to choice of a particular three-way grouping of the factor-level combinations.

The final questions of interest culminated in the form "During software development, what comparisons between the effects of the three factor-level combinations (a) single individuals, (b) ad hoc teams, and (c) disciplined teams appear as differences in the various quantitatively measurable aspects of the software development process and product? Furthermore, what kind of differences are exhibited and what is the direction of these differences?"

Diagram 1. Approach Schematic



## Step 2: Research Hypotheses

Since the investigative methodology involves hypothesis testing, it is necessary to have fairly precise statements, called research hypotheses, which are to be either supported or refuted by the evidence. The second step in the approach was to formulate these research hypotheses, disjoint pairs designated null and alternative, from the questions of interest.

A precise meaning was given to the notion "what kind of difference." The investigation considered both (a) differences in central tendency or average value, and (b) differences in variability around the central tendency, of observed values of the quantifiable programming aspects. It should be noted that this decision to examine both location and dispersion comparisons among the experimental groups brought a pervasive duality to the entire investigation (i.e., two sets of statistical tests, two sets of statistical results, two sets of conclusions, etc. --always in parallel and independent of each other--), since it addresses both the expectancy and the predictability of behavior under the experimental treatments.

Some vagueness was removed regarding the size of the particular programming task by making explicit the implicit restriction that completion of the task not be beyond the capability of a single programmer working alone for a reasonable period of time. Additionally, a large set of programming aspects were specified; they are discussed in Section II, Specifics. For each programming aspect there were similar questions of interest, similar research hypotheses and similar experiments conducted in parallel.

The schema for the research hypotheses may be stated as "In the context of a one-person do-able software development project, there < is not | is > a difference in the < location | dispersion > of the measurements on programming aspect < X > between individuals (AI), ad hoc teams (AT), and disciplined teams

(DT)." For each programming aspect 'X' in the set under consideration, this schema generates two pairs of nondirectional research hypotheses, depending upon the selection of 'is not' or 'is' corresponding to the null and alternative hypotheses, and the selection of 'location' or 'dispersion' corresponding to the type of difference.

### Step 3: Statistical Model

The choice of a statistical model makes explicit various assumptions regarding the experimental design, such as the dependent variables observed, the distributions of the underlying populations, etc. Because the study involves a homogeneity-of-populations problem with shift and spread alternatives, the multi-sample model used here requires the following criteria: independent populations, independent and random sampling within each population, and interval scale of measurement [Conover 71; pp. 65-67] for each programming aspect. Although random sampling was not explicitly achieved in this study by rigorous sampling procedures, it was nonetheless assumed on the basis of the apparent representativeness of the subject pool and the lack of obvious reasons to doubt otherwise. Due to the small sample sizes, the unknown shape of the underlying distributions, and the partially exploratory nature of the study, a nonparametric statistical model was used.

Whenever statistics is employed to "prove" that some systematic effect --in this case, a difference among the groups-- exists, it is important to measure the risk of error. This is usually done by reporting a significance level  $\alpha$  [Conover 71; p. 79], which represents the probability of deciding that a systematic effect exists when in fact it does not. In the model, the hypothesis testing for each programming aspect was regarded as a separate independent experiment. As a consequence of this choice, the significance level is controlled and reported experimentwise (i.e., on a per aspect basis). While the assumption of independence between such experiments is not

entirely supportable, this procedure is valid as long as conclusions that couple one or more of these programming aspects are avoided or properly qualified. In this study, statements regarding interrelationships among aspects are made only within the interpretations in Section V.

#### Step 4: Statistical Hypotheses

The research hypotheses must be translated into statistically tractable form, called statistical hypotheses. A correspondence, governed by the statistical model, exists between application-oriented notions in the research hypotheses (e.g., typical performance of a programming team under the disciplined methodology) and mathematical notions in the statistical hypotheses (e.g., expected value of a random variable defined over the population from which the disciplined teams are a representative sample). Generally speaking, only certain mathematical statements involving pairs of populations are statistically tractable, in the sense that standard statistical procedures are applicable. Statements that are not directly tractable may be broken down into tractable (sub)components whose results are properly recombined after having been decided individually.

In this study, the research hypotheses are concerned with directional differences among three programming environments. Since the corresponding mathematical statements are not directly tractable, they were broken down into the set of seven statistical hypotheses pairs shown below. The hypotheses pair

null:  $AI = AT = DT$       alternative:  $-(AI = AT = DT)$

addresses the existence of an overall difference among the groups. However, due to the weak nondirectional alternative, it cannot indicate which groups are different or in what direction a difference lies. Standard statistical practice prescribes that a successful test for overall difference among three or more groups be followed by tests for pairwise differences. The hypotheses pairs

null: AI = AT	alternative: AI $\neq$ AT or
null: AT = DT	alternative: AI < AT or AT < AI
null: AI = DT	alternative: AT $\neq$ DT or
	alternative: AT < DT or DT < AT
	alternative: AI $\neq$ DT or
	alternative: AI < DT or DT < AI

address the existence and direction of pairwise differences between groups. The results of these pairwise comparisons were used to explicate the overall comparison. Data collected for a set of experiments may often be legitimately reused to "simulate" other closely related experiments, by combining certain samples together and ignoring the original distinction(s) between them. It is meaningful, in the context of this study's experimental design, to compare any two groups pooled against the third since (1) AI and AT are both undisciplined, while DT is disciplined; (2) AT and DT are both teams, and AI is individuals; and (3) under the assumption that disciplined teams behave like individuals --which is part of the study's basic premise--, DT and AI can be pooled and compared with AT acting as a control group. The hypotheses pairs

null: AI+AT = DT	alternative: AI+AT $\neq$ DT or
null: AT+DT = AI	alternative: AI+AT < DT or DT < AI+AT
null: AI+DT = AT	alternative: AT+DT $\neq$ AI or
	alternative: AT+DT < AI or AI < AT+DT
	alternative: AI+DT $\neq$ AT or
	alternative: AI+DT < AT or AT < AI+DT

address the existence and direction of such pooled differences. The results of these pooled comparisons were used to corroborate the overall and pairwise comparisons.

Thus, for any particular programming aspect, the research hypotheses pair corresponds to seven different pairs (null and alternative) of scientific hypotheses. The results of testing each set of seven hypotheses must be abstracted and organized into one statistical conclusion using the first research framework discussed in the next step.

#### Step 5: Research Frameworks

The research frameworks provide the necessary organizational basis for abstracting and conceptualizing the massive volume of statistical hypotheses (and statistical results that follow) into

a smaller and more intellectually manageable set of conclusions. Three separate research frameworks have been chosen: (1) the framework of possible overall comparison outcomes for a given programming aspect, (2) the framework of dependencies and intuitive relationships among the various programming aspects considered, and (3) the framework of basic suppositions regarding expected effects of the experimental treatments on the comparison outcomes for the entire set of programming aspects. The first framework is employed in the statistical conclusions step because it can be applied in a statistically tractable manner, while the remaining two frameworks are reserved for employment in the research interpretations step since they are not statistically tractable and involve subjective judgement.

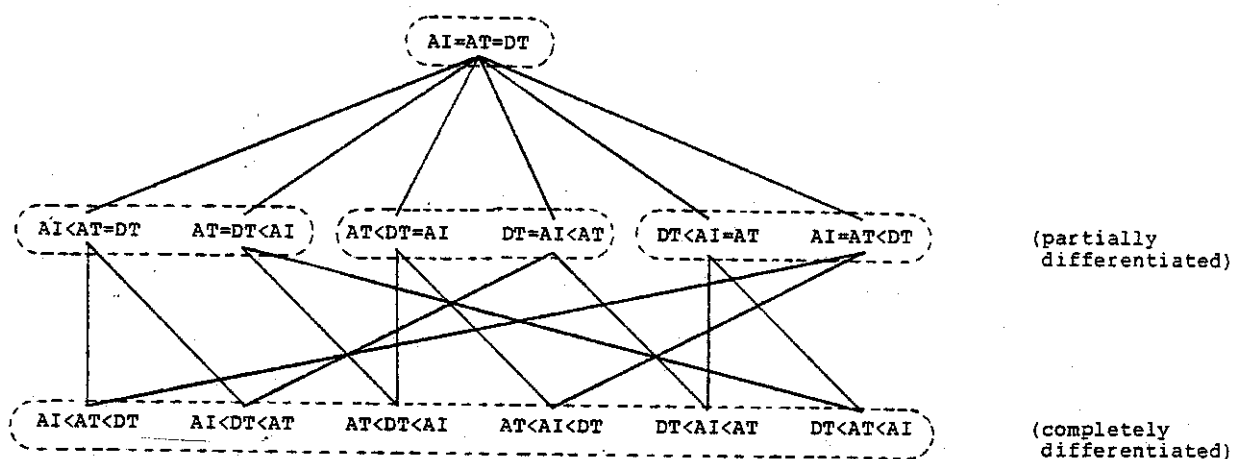
Since a finite set of three different programming environments (AI, AT, and DT) are being compared, there exists the following finite set of thirteen possible overall comparison outcomes for each aspect considered:

$$\begin{array}{l}
 \text{AI} = \text{AT} = \text{DT} \\
 \left. \begin{array}{l} \text{AI} < \text{AT} = \text{DT} \\ \text{AT} = \text{DT} < \text{AI} \end{array} \right\} \text{AI} \neq \text{AT} = \text{DT} \\
 \left. \begin{array}{l} \text{AT} < \text{DT} = \text{AI} \\ \text{DT} = \text{AI} < \text{AT} \end{array} \right\} \text{AT} \neq \text{DT} = \text{AI} \\
 \left. \begin{array}{l} \text{DT} < \text{AI} = \text{AT} \\ \text{AI} = \text{AT} < \text{DT} \end{array} \right\} \text{DT} \neq \text{AI} = \text{AT} \\
 \left. \begin{array}{l} \text{AI} < \text{AT} < \text{DT} \\ \text{AI} < \text{DT} < \text{AT} \\ \text{AT} < \text{DT} < \text{AI} \\ \text{AT} < \text{AI} < \text{DT} \\ \text{DT} < \text{AI} < \text{AT} \\ \text{DT} < \text{AT} < \text{AI} \end{array} \right\} \text{AI} \neq \text{AT} \neq \text{DT}
 \end{array}$$

There is a hierarchical lattice of increasing separation and directionality among these possible overall comparison outcomes as shown in Diagram 2. These thirteen possible-overall comparison outcomes comprise the first research framework and may be viewed as providing a complete "answer space" for the questions of interest. It is clear that any consistent set of two-way comparisons (such as represented in the statistical hypotheses or statistical results) may be associated with a unique one of these three-way comparisons. This framework is the basis for organizing and condensing the seven statistical results into one statistical conclusion for each programming aspect considered.

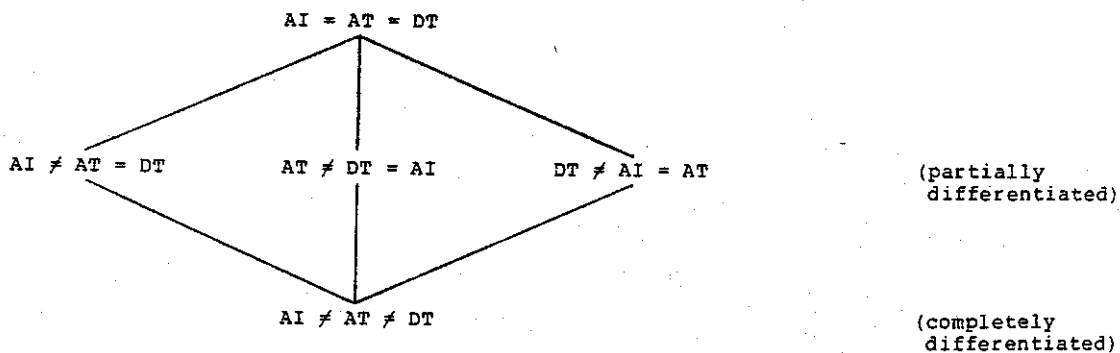


Diagram 2.1 Lattice of Possible Directional Outcomes for Three-way Comparison



N.B. The circles indicate which directional outcomes correspond to the same nondirectional outcome.

Diagram 2.2 Lattice of Possible Nondirectional Outcomes for Three-way Comparison



Since a large set of interrelated programming aspects are being examined, it would be desirable to summarize many of the "per aspect" hypotheses and results into statements which refer to several aspects simultaneously. For example, average number of statements per segment is one aspect directly dependent on two other aspects: number of segments and number of statements. Other interrelationships are more intuitive, less tractable, or only suspected, for example, the "trade-off" between global variables and formal parameters. A simple classification of the programming aspects into groups of intuitively related aspects at least provides a framework for jointly interpreting the corresponding statistical conclusions in light of the underlying issues by which the aspects themselves are related. The programming aspects considered in this study were classified according to a particular set of nine higher-level programming issues (such as data variable organization, for example); details are given in Section V, Interpretive Results. This second research framework is the basis for abstracting and interpreting what the study's findings indicate about these higher-level programming issues, as well as explicitly mentioning several individual relationships among the programming aspects and their conclusions.

Since the design of the experiments, the choice of treatment factors, etc., were at least partially motivated by certain general beliefs regarding software development (such as "disciplined methodology reduces software development costs", for example), it should be possible to explicitly state what comparison outcomes among the experimental treatments were expected a priori for which programming aspects. A list of preplanned expectations (so-called "basic suppositions") for the outcomes of each aspect's experiment would provide a framework for evaluating how well the experimental findings as a whole support the underlying general beliefs (by comparing the actual outcomes with the basic suppositions across all the programming aspects). Such a list of basic suppositions was conceived prior to conducting the experiments, and it constitutes the third research framework; details are given in Section V, Interpretive Results.

This framework is the basis for interpreting the study's findings in terms of evidence in favor of the basic suppositions and general beliefs.

#### Step 6: Experimental Design

The experimental design is the plan or setup according to which the experiment is actually conducted or executed. It is based upon the statistical model, and deals with practical issues such as experimental units, treatment factors and levels, experimental local control, etc. The experimental design employed for this study has been discussed in considerable detail in Section II, Specifics.

#### Step 7: Collected Data

The pertinent data to carry out the experimental design was collected and processed to yield the information to which the statistical test procedures were applied. Some details of this execution phase are given in Section II, Specifics.

#### Step 8: Statistical Test Procedures

A statistical test procedure is a decision mechanism, founded upon general principles of mathematical probability and combinatorics and upon a specific statistical model (i.e., requiring certain assumptions), which is used to convert the statistical hypotheses together with the collected data into the statistical results. As dictated by the statistical model, the statistical tests used in the study were nonparametric tests of homogeneity of populations against shift alternatives for small samples. Nonparametric tests are slightly more conservative (in rejecting the null hypothesis) than their parametric counterparts; nonparametric tests generally use the ordinal ranks associated with a linear ordering of a set of scores, rather than the scores themselves, in their computational formulas. In particular, the standard Kruskal-Wallis H-test [Siegel 56; pp. 184-193] and

Mann-Whitney U-test [Siegel 56; pp. 116-127] were employed in the statistical results step. Ryan's Method of Adjusted Significance Levels [Kirk 68; pp. 97, 495-497], a standard procedure for controlling the experimentwise significance level when several tests are performed on the same scores as one experiment, was also employed in the statistical conclusions step.

The Kruskal-Wallis test is used in three-sample situations to test an  $X = Y = Z$  null hypothesis; its test statistic is computed as

$$H = 12 * [(R_x * R_x / n_x) + (R_y * R_y / n_y) + (R_z * R_z / n_z)] / [(n) * (n+1)] - 3 * (n+1)$$

where  $R_x$ ,  $R_y$ , and  $R_z$  are the respective sums of the ranks for scores from the X, Y, and Z samples;  $n$  equals  $n_x + n_y + n_z$ ; and  $n_x$ ,  $n_y$ , and  $n_z$  are the respective sample sizes. The Mann-Whitney test is used in two-sample situations to test an  $X = Y$  null hypothesis; its test statistic is computed as

$$U = \min [ n_x * n_y + n_x * (n_x + 1) / 2 - R_x ; n_y * n_x + n_y * (n_y + 1) / 2 - R_y ]$$

where  $R_x$ ,  $R_y$ ,  $n_x$ , and  $n_y$  are defined as before.

For every statistical test, there exists a one-to-one mapping, usually given in statistical tables, between the test statistic --whose value is completely determined by the sample data scores-- and the critical level. The critical level  $\hat{\alpha}$  [Conover 71; p. 81] is defined as the minimum significance level at which the statistical test procedure would allow the null hypothesis to be rejected (in favor of the alternative) for the given sample data. Thus critical level represents a concise standardized way to state the full result of any statistical test procedure. Two-tailed rejection regions are applied for tests involving nondirectional alternative hypotheses, and one-tailed rejection regions are applied for tests involving directional alternative hypotheses, so that the stated critical level always pertains directly to the stated alternative hypothesis. A decision to reject the null hypothesis and accept the alternative is mandated if the critical level is low enough to be tolerated; otherwise a decision to retain the null hypothesis is made.

The Ryan's procedure is used in situations involving multiple pairwise comparisons, in order to properly account for the fact that each pairwise test is made in conjunction with the others, using the same sample data. The individual critical levels  $\hat{\alpha}$  obtained for each pairwise test in isolation are adjusted to proper experimentwise critical levels  $\hat{\alpha}'$  via the formula

$$\hat{\alpha}' = [(r+1)*k/2] * \hat{\alpha}$$

where  $k$  is the total number of samples; and  $r$  is the number of (other) samples whose rank means fall between the rank means of the particular pair of samples being compared. A simple "minimax" step --taking the maximum of the several adjusted pairwise critical levels, plus the overall comparison critical level, which are all minimum significance levels-- completes the procedure, yielding a single critical level associated jointly with the overall and pairwise comparisons.

These tests and procedures apply straightforwardly when differences in location are considered. A slight modification makes them applicable for differences in dispersion: prior to ranking, each score value is simply replaced by its absolute deviation from the corresponding within-group sample median [Nemenyi et al. 77; pp. 266-270]. It should be noted that this modification results in only an approximate method for solving a tough statistical problem, namely, testing whether one population is more variable than another [Nemenyi et al. 77; pp. 279-283]. The modification is not strictly statistically "kosher" in the general case (it weakens the power of the test procedures and can yield inaccurate critical levels when testing for dispersion differences), but every other available method also has serious limitations. This method has been shown to possess reasonable accuracy as long as the underlying distributions are fairly symmetrical and it adapts easily to the study's three-way comparison situation.

#### Step 9: Statistical Results

A statistical result is essentially a decision reached by

applying a statistical test procedure to the set of collected and refined data, regarding which one of the corresponding pair (null, alternative) of statistical hypotheses is indeed supported by that data. For each pair of statistical hypotheses, there is one statistical result consisting of four components: (1) the null hypothesis itself; (2) the alternative hypothesis itself; (3) the critical level, stated as a probability value between 0 and 1; and (4) a decision either to retain the null hypothesis or to reject it in favor of (i.e., accept) the alternative hypothesis.

By convention, the null hypothesis purports that no systematic difference appears to exist, and the alternative hypothesis purports that some systematic difference seems to exist. The critical level is associated with erroneously accepting the alternative hypothesis (i.e., claiming a systematic difference when none in fact exists). The decision to retain or reject is reached on the basis of some tolerable level of significance, with which the critical level is compared to see if it is low enough. In cases where a null hypothesis is rejected, the appropriate directional alternative hypothesis (if any) is used to indicate the direction of the systematic difference, as determined by direct observation from the sample medians in conjunction with a one-tailed test.

Conventional practice is to fix an arbitrary significance level (e.g., .05 or .01) in advance, to be used as the tolerable level; critical levels then serve only as stepping-stones toward reaching decisions and are not reported. For this partially exploratory study, it was deemed more appropriate to fix a tolerable level only for the purpose of a screening decision (which simply purges those results with intolerably high critical levels), and to carry the actual critical level along with each statistical result. This unconventional practice yields statistical results in a more meaningful and flexible form, since the significance or error risk of each result may be assessed individually, and results at other more stringent significance levels may be easily determined. Furthermore, the necessary

information is retained for properly recombining multiple related results on an experimentwise basis in the statistical conclusions step.

The tolerable level of significance used throughout this study to screen critical levels was fixed at under .20. Although fairly high for a confirmatory study, it is reasonable for a partially exploratory study, such as this one, seeking to discover even slight trends in the data. A critical level of .20 means that the odds of obtaining test scores exhibiting the same degree of difference, due to random chance fluctuations alone, are one in five.

As an example, the seven statistical results for location comparisons on the programming aspect STATEMENT TYPE COUNTS\IF are shown below. (N.B. The asterisks will be explained in Step 10.)

null hypothesis	alternative hypothesis	critical level	(screening) decision
AI = AT = DT	-(AI = AT = DT)	.0630	reject
AI = AT	AI < AT	.0465	reject
AI = DT	AI ≠ DT	>.9999	retain
AT = DT	DT < AT	.0111	reject
AI+AT = DT	DT < AI+AT	.0884	reject *
AI+DT = AT	AI+DT < AT	.0029	reject
AT+DT = AI	AT+DT ≠ AI	.3352	retain *

Observe that the stated decisions simply reflect the application of the .20 tolerable level to the stated critical levels. Results under more stringent levels of significance can be easily determined by simply applying a lower tolerable level to form the decisions; e.g., at the .05 significance level, only the AI < AT, DT < AT, and AI+DT < AT alternative hypotheses would be accepted; only the AI+DT < AT hypothesis would be accepted at the .01 level.

### Step 10: Statistical Conclusions

The volume of statistical results are organized and condensed into statistical conclusions according to the prearranged research framework(s). A statistical conclusion is an abstraction of several statistical results, but it retains the same statistical character, having been derived via statistically tractable methods and possessing an associated critical level.

Specifically, the first research framework mentioned above was employed to reduce the seven statistical results (with seven individual critical levels) for each programming aspect to a single statistical conclusion (with one overall critical level) for that aspect. The statement portion of a statistical conclusion is simply one of the thirteen possible overall comparison outcomes. Each overall comparison outcome is associated with a particular set of statistical results whose outcomes support the overall comparison outcome in a natural way. For example, the  $DT = AI < AT$  conclusion is associated with the following results:

- reject  $AI = AT = DT$  in favor of  $-(AI = AT = DT)$ ,
- reject  $AI = AT$  in favor of  $AI < AT$ ,
- retain  $AI = DT$ ,
- reject  $AT = DT$  in favor of  $DT < AT$ , and
- reject  $AI+DT = AT$  in favor of  $AI+DT < AT$ .

Since the other two comparisons ( $AI+AT$  versus  $AT$ ,  $AT+DT$  versus  $AI$ ) are in a sense orthogonal to the overall comparison outcome ( $DT = AI < AT$ ), their results are considered irrelevant to this conclusion. The chart in Diagram 3 shows exactly which results are associated with each conclusion: the relevant comparisons, the null hypotheses to be retained, and the alternative hypotheses to be accepted. The other portion of a statistical conclusion is the critical level associated with erroneously accepting the statement portion. It is computed from the individual critical levels of certain germane results.

A simple deterministic algorithm, based on the chart in Diagram 3, was used to generate the statistical conclusions (and compute the overall critical level) automatically from the statistical results. For each programming aspect, the algorithm compared the actual results obtained for the seven statistical hypotheses pairs with the results associated with each conclusion, searching for a match. Ryan's procedure was used to properly combine the individual critical levels for the overall result and the relevant pairwise results, by adjusting them via the formula and then taking their maximum. The critical levels for the



Diagram 3. Association Chart for Results and Conclusions

The following chart specifies which statistical results are associated with each statistical conclusion. An asterisk indicates a comparison not relevant to that conclusion; the null hypothesis appears wherever it must be retained; and the alternative hypothesis to be accepted appears wherever the null hypothesis must be rejected. If a set of results do not satisfy the criteria associated with any of the non-null conclusions, then they are associated with the null conclusion by default.

conclusions:	AI=AT=DT	AI=AT	AI=DT	AT=DT	AI+AT=DT	AI+DT=AT	AT+DT=AI
AI = AT = DT							
AI < AT = DT					*	*	AI<AT+DT
AT = DT < AI					*	*	AT+DT<AI
AT < DT = AI					*	AT<AI+DT	*
DT = AI < AT					*	AI+DT<AT	*
DT < AI = AT							*
AI = AT < DT							*
AI < AT < DT							*
AI < DT < AT							AI<AT+DT
AT < DT < AI					*	AI+DT<AT	AI<AT+DT
AT < AI < DT					*	AT<AI+DT	AT+DT<AI
DT < AI < AT						AT<AI+DT	*
DT < AT < AI							*

results:

[default]

relevant pooled results were factored in via a simple formula based on the multiplicative rule for the joint probability of independent events.

Continuing the example started in Step 9, the statistical results shown there for location comparisons on the STATEMENT TYPE COUNTS\IF aspect are reduced to the statistical conclusion  $DT = AI < AT$  with .0780 critical level overall. The five results not marked with an asterisk in Step 9 match the five results associated above with the  $DT = AI < AT$  outcome. (Note that the other two marked results represent comparisons which are irrelevant to this conclusion.) The .0465 and .0111 critical levels for the two pairwise differences are adjusted to .0697 and .0332, and the maximum of those adjusted values plus the .0630 overall difference critical level is .0697. The relevant pooled comparison critical level of .0089 is factored in by taking the complement of the products of the complements:

$$1 - [(1 - .0697) * (1 - .0089)] = .0780$$

Thus, the statistical conclusions are in one-to-one correspondence with the research hypotheses and provide concise answers on a "per aspect" basis to the questions of interest. Further details and complete listing of the statistical conclusions for this study are presented below in Section IV.

#### Step 11: Research Interpretations

The final step in the approach is to interpret the statistical conclusions in view of any remaining research framework(s), the researchers' intuitive and professional expectations, and the work of other researchers. These research interpretations provide the opportunity to augment the objective findings of the study with the researcher's own subjective judgments and interpretations. The second and third research frameworks mentioned above --namely, the intuitive relationships among the various programming aspects and the basic suppositions governing their expected outcomes-- were considered important.

However these particular research frameworks can only be utilized for the research interpretations, since they are not amenable to rigorous manipulation. Nonetheless, within these frameworks which are based upon intuitive understanding about the programming aspects and software development environments under consideration, the study bears some of its most interesting results and implications. Complete details and discussion of the research interpretations of this study appear in Section V.

#### IV. Objective Results

This section reports the objective results of the study, namely, the statistical conclusions for each programming aspect considered. The tone of discussion here is purposely somewhat disinterested and analytical, in keeping with the empirical and statistical character of these conclusions. All interpretive discussion is deferred to Section V.

Each statistical conclusion is expressed in the concise form of a three-way comparison outcome "equation." It states any observed differences, and the directions thereof, among the programming environments represented by the three groups examined in the study: ad hoc individuals (AI), ad hoc teams (AT), and disciplined teams (DT). The equality  $AI = AT = DT$  expresses the null conclusion that there is no systematic difference among the groups. An inequality, e.g.,  $AI < AT = DT$  or  $DT < AI < AT$ , expresses a non-null (or alternative) conclusion that there are certain systematic difference(s) among the groups in stated direction(s). A critical level (or risk) value is also associated with each non-null (or alternative) conclusion, indicating its individual reliability. This value is the probability of having erroneously rejected the null conclusion in favor of the alternative; it also provides a relative index of how pronounced the differences were in the sample data.

The remainder of this section consists of (a) presenting the full set of conclusions, (b) evaluating their impact as a whole, (c) exposing a "relaxed differentiation" view of the conclusions, (d) exposing a "directionless" view of the conclusions, and (e) individually highlighting a few of the more noteworthy conclusions.

#### Presentation

Instances of non-null (or alternative) conclusions indicating

some distinction among the groups on the basis of a particular programming aspect are listed by outcome in Tables 2.1 (for location comparisons) and 2.2 (for dispersion comparisons). A complete itemization of these distinctions, in English prose form, appears in Appendix 2. The complete set of statistical conclusions for both location and dispersion comparisons appears in Table 3 arranged by programming aspect.

Examination of Table 3 immediately demonstrates that a large number of the programming aspects considered in this study, especially product aspects, failed to show any distinction between the groups. This low "yield" is not surprising, especially among product aspects, and may be attributed to the partially exploratory nature of the study, the small sample sizes, and the general coarseness of many of the aspects considered. The issue of these null outcome occurrences and their significance is treated more thoroughly in the next subsection, Impact Evaluation.

It is worth noting, however, that several of the null conclusions may indicate characteristics inherent to the application itself. As one example, the basic symbol-table/scan/parse/code-generation nature of a compiler strongly influences the way the system is modularized and thus practically determines the number of modules in the final product (give or take some occasional slight variation due to other design decisions).

### Impact Evaluation

These statistical conclusions have a certain objective character --since they are statistically inferred from empirical data-- and their collective impact may be objectively evaluated according to the following statistical principle [Tukey 69; p. 84-85]. Whenever a series of statistical tests (or experiments) are made, all at a fixed level of significance (for example, .10), a corresponding percentage (in the example, 10%) of the tests are expected a priori to reject the null hypothesis in the complete absence of any true effect (i.e., due to chance alone). This

Table 2.1 Non-Null Conclusions, for Location Comparisons, arranged by outcome

comparison outcome	critical level *	programming aspect
AI < AT = DT	8	SEGMENTS DATA VARIABLES SCOPE COUNTS \ GLOBAL DATA VARIABLE SCOPE COUNTS \ NONGLOBAL \ MODIFIED DATA VARIABLE SCOPE COUNTS \ NONGLOBAL \ PARAMETER DATA VARIABLE SCOPE PERCENTAGES \ NONGLOBAL \ PARAMETER AVERAGE NONGLOBAL VARIABLES PER SEGMENT \ PARAMETER (SEG,GLOBAL) POSSIBLE USAGE PAIRS
AT = DT < AI	5	AVERAGE STATEMENTS PER SEGMENT AVG INVOCATIONS PER (CALLING) SEGMENT \ NONINTRINSIC AVG INVOCATIONS PER (CALLED) SEGMENT AVG INVOCATIONS PER (CALLED) SEGMENT \ FUNCTION DATA VARIABLE SCOPE PERCENTAGES \ NONGLOBAL \ LOCAL (SEG,GLOBAL) USAGE RELATIVE PERCENTAGES \ NONENTRY \ UNMODIFIED
AT < DT = AI	1	
DT = AI < AT	8	STATEMENT TYPE COUNTS \ IF STATEMENT TYPE COUNTS \ (PROC)CALL \ INTRINSIC STATEMENT TYPE COUNTS \ RETURN STATEMENT TYPE PERCENTAGES \ IF DECISIONS INVOCATIONS \ PROCEDURE \ INTRINSIC INVOCATIONS \ INTRINSIC (SEG,GLOBAL,SEG) DATA BINDINGS \ POSSIBLE
DT < AI = AT	8	COMPUTER JOB STEPS \ MODULE COMPILATIONS \ UNIQUE COMPUTER JOB STEPS \ MODULE COMPILATIONS COMPUTER JOB STEPS \ PROGRAM EXECUTIONS COMPUTER JOB STEPS \ MISCELLANEOUS ESSENTIAL JOB STEPS AVERAGE UNIQUE COMPILATIONS PER MODULE MAX UNIQUE COMPILATIONS F.A.O. MODULE
AI = AT < DT	0	
AI < AT < DT	0	
AI < DT < AT	1	LINES
AT < DT < AI	2	(SEG,GLOBAL) USAGE RELATIVE PERCENTAGES \ ENTRY \ MODIFIED (SEG,GLOBAL) USAGE RELATIVE PERCENTAGES \ ENTRY \ MODIFIED
AT < AI < DT	0	
DT < AI < AT	1	PROGRAM CHANGES
DT < AT < AI	0	

\* this column records the frequency of occurrence for each comparison outcome

Table 2.2 Non-Null Conclusions, for Dispersion Comparisons, arranged by outcome

comparison outcome	critical level *	programming aspect
AI < AT = DT	6	DATA VARIABLE SCOPE COUNTS \ NONGLOBAL \ PARAMETER \ REFERENCE DATA VARIABLE SCOPE PERCENTAGES \ NONGLOBAL \ PARAMETER \ REFERENCE PARAMETER PASSAGE TYPE PERCENTAGES \ VALUE PARAMETER PASSAGE TYPE PERCENTAGES \ REFERENCE (SEG,GLOBAL) POSSIBLE USAGE PAIRS \ NONENTRY (SEG,GLOBAL,SEG) DATA BINDINGS \ ACTUAL \ INDEPENDENT
AT = DT < AI	3	COMPUTER JOB STEPS \ MISCELLANEOUS INVOCATIONS INVOCATIONS \ NONINTRINSIC
AT < DT = AI	3	STATEMENTS AVG INVOCATIONS PER (CALLED) SEGMENT \ FUNCTION (SEG,GLOBAL) ACTUAL USAGE PAIRS \ MODIFIED
DT = AI < AT	5	AVERAGE SEGMENTS PER MODULE STATEMENT TYPE PERCENTAGES \ RETURN AVERAGE GLOBAL VARIABLES PER MODULE \ MODIFIED (SEG,GLOBAL) POSSIBLE USAGE PAIRS \ NONENTRY \ MODIFIED (SEG,GLOBAL,SEG) DATA BINDINGS \ POSSIBLE
DT < AI = AT	6	STATEMENT TYPE COUNTS \ (PROC)CALL \ NONINTRINSIC STATEMENT TYPE COUNTS \ (PROC)CALL \ NONINTRINSIC INVOCATIONS \ PROCEDURE INVOCATIONS \ PROCEDURE \ NONINTRINSIC AVG INVOCATIONS PER (CALLING) SEGMENT \ PROCEDURE \ INTRINSIC DATA VARIABLE SCOPE PERCENTAGES \ GLOBAL \ NONENTRY \ MODIFIED
AI = AT < DT	7	AVERAGE TOKENS PER STATEMENT DATA VARIABLE SCOPE COUNTS \ GLOBAL DATA VARIABLE SCOPE COUNTS \ NONGLOBAL \ PARAMETER DATA VARIABLE SCOPE PERCENTAGES \ GLOBAL DATA VARIABLE SCOPE PERCENTAGES \ NONGLOBAL DATA VARIABLE SCOPE PERCENTAGES \ NONGLOBAL \ PARAMETER \ VALUE DATA VARIABLE SCOPE PERCENTAGES \ NONGLOBAL \ PARAMETER \ VALUE
AI < AT < DT	0	(SEG,GLOBAL) POSSIBLE USAGE PAIRS \ NONENTRY \ UNMODIFIED (SEG,GLOBAL) POSSIBLE USAGE PAIRS \ NONENTRY \ UNMODIFIED
AI < DT < AT	0	
AT < DT < AI	0	
AT < AI < DT	2	
DT < AT < AI	0	MAX UNIQUE COMPILATIONS F.A.O. MODULE STATEMENT TYPE COUNTS \ RETURN

\* this column records the frequency of occurrence for each comparison outcome

Table 3. Statistical Conclusions

N.B. A simple pair of equal signs ( = = ) appears in place of the null outcome AI = AT = DT in order to avoid cluttering the table excessively.

programming aspect	location comparison :critical outcome : level	dispersion comparison :critical outcome : level
*****		
development process aspects :	:	:
=====	=====	=====
COMPUTER JOB STEPS	DT < AI = AT : 0.0036	= = :
MODULE COMPILATIONS	DT < AI = AT : 0.0223	= = :
UNIQUE	DT < AI = AT : 0.0110	= = :
IDENTICAL	= = :	= = :
PROGRAM EXECUTIONS	DT < AI = AT : 0.0221	= = :
MISCELLANEOUS	DT < AI = AT : 0.1445	AT = DT < AI : 0.0775
=====	=====	=====
ESSENTIAL JOB STEPS	DT < AI = AT : 0.0037	= = :
AVERAGE UNIQUE COMPILATIONS PER MODULE	DT < AI = AT : 0.0883	= = :
MAX UNIQUE COMPILATIONS F.A.O. MODULE	DT < AI = AT : 0.1180	DT < AI < AT : 0.0514
=====	=====	=====
PROGRAM CHANGES	DT < AI < AT : 0.1848	= = :
*****		
final product aspects :	:	:
=====	=====	=====
MODULES	= = :	= = :
=====	=====	=====
SEGMENTS	AI < AT = DT : 0.0634	= = :
=====	=====	=====
SEGMENT TYPE COUNTS :	:	:
FUNCTION	= = :	= = :
PROCEDURE	= = :	= = :
=====	=====	=====
SEGMENT TYPE PERCENTAGES :	:	:
FUNCTION	= = :	= = :
PROCEDURE	= = :	= = :
=====	=====	=====
AVERAGE SEGMENTS PER MODULE	= = :	DT = AI < AT : 0.0218
=====	=====	=====
LINES	AI < DT < AT : 0.1194	= = :
=====	=====	=====
STATEMENTS	= = :	AT < DT = AI : 0.1954
=====	=====	=====
STATEMENT TYPE COUNTS :	:	:
:=	= = :	= = :
IF	DT = AI < AT : 0.0780	= = :
CASE	= = :	= = :
WHILE	= = :	= = :
EXIT	= = :	= = :
(PROC)CALL	= = :	DT < AI = AT : 0.0325
NONINTRINSIC	= = :	DT < AI = AT : 0.1862
INTRINSIC	DT = AI < AT : 0.1732	= = :
RETURN	DT = AI < AT : 0.0860	DT < AI < AT : 0.1398
=====	=====	=====
STATEMENT TYPE PERCENTAGES :	:	:
:=	= = :	= = :
IF	DT = AI < AT : 0.1069	= = :
CASE	= = :	= = :
WHILE	= = :	= = :
EXIT	= = :	= = :
(PROC)CALL	= = :	= = :
NONINTRINSIC	= = :	= = :
INTRINSIC	= = :	= = :
RETURN	= = :	DT = AI < AT : 0.0401
=====	=====	=====
AVERAGE STATEMENTS PER SEGMENT	AT = DT < AI : 0.1706	= = :
=====	=====	=====
AVERAGE STATEMENT NESTING LEVEL	= = :	= = :
=====	=====	=====
DECISIONS	DT = AI < AT : 0.1468	= = :
=====	=====	=====



FUNCTION CALLS	= = :	= = :
NONINTRINSIC	= = :	= = :
INTRINSIC	= = :	= = :
TOKENS	= = :	= = :
AVERAGE TOKENS PER STATEMENT	= = :	AI = AT < DT : 0.1061
INVOCATIONS	= = :	AT = DT < AI : 0.0206
FUNCTION	= = :	= = :
NONINTRINSIC	= = :	= = :
INTRINSIC	= = :	= = :
PROCEDURE	= = :	DT < AI = AT : 0.0325
NONINTRINSIC	DT = AI < AT : 0.1732	DT < AI = AT : 0.1862
INTRINSIC	= = :	AT = DT < AI : 0.0510
NONINTRINSIC	DT = AI < AT : 0.0435	= = :
INTRINSIC	= = :	= = :
AVG INVOCATIONS PER (CALLING) SEGMENT	= = :	= = :
FUNCTION	= = :	= = :
NONINTRINSIC	= = :	= = :
INTRINSIC	= = :	= = :
PROCEDURE	= = :	= = :
NONINTRINSIC	= = :	= = :
INTRINSIC	= = :	DT < AI = AT : 0.0653
NONINTRINSIC	AT = DT < AI : 0.1699	= = :
INTRINSIC	= = :	= = :
AVG INVOCATIONS PER (CALLED) SEGMENT	AT = DT < AI : 0.1699	= = :
FUNCTION	AT = DT < AI : 0.1936	AT < DT = AI : 0.1411
PROCEDURE	= = :	= = :
DATA VARIABLES	AI < AT = DT : 0.0698	= = :
DATA VARIABLE SCOPE COUNTS :		
GLOBAL	AI < AT = DT : 0.1476	AI = AT < DT : 0.1241
ENTRY	= = :	= = :
MODIFIED	= = :	= = :
UNMODIFIED	= = :	= = :
NONENTRY	= = :	= = :
MODIFIED	= = :	= = :
UNMODIFIED	= = :	= = :
MODIFIED	AI < AT = DT : 0.1614	= = :
UNMODIFIED	= = :	= = :
NONGLOBAL	= = :	= = :
PARAMETER	AI < AT = DT : 0.1271	AI = AT < DT : 0.1061
VALUE	= = :	= = :
REFERENCE	= = :	AI < AT = DT : 0.0199
LOCAL	= = :	= = :
DATA VARIABLE SCOPE PERCENTAGES :		
GLOBAL	= = :	AI = AT < DT : 0.0750
ENTRY	= = :	= = :
MODIFIED	= = :	= = :
UNMODIFIED	= = :	= = :
NONENTRY	= = :	= = :
MODIFIED	= = :	DT < AI = AT : 0.0218
UNMODIFIED	= = :	= = :
MODIFIED	= = :	= = :
UNMODIFIED	= = :	= = :
NONGLOBAL	= = :	AI = AT < DT : 0.0750
PARAMETER	AI < AT = DT : 0.1507	AI = AT < DT : 0.0557
VALUE	= = :	AI = AT < DT : 0.0943
REFERENCE	= = :	AI < AT = DT : 0.1529
LOCAL	AT = DT < AI : 0.1090	= = :
AVERAGE GLOBAL VARIABLES PER MODULE	= = :	= = :
ENTRY	= = :	= = :
NONENTRY	= = :	= = :
MODIFIED	= = :	DT = AI < AT : 0.1100
UNMODIFIED	= = :	= = :
AVERAGE NONGLOBAL VARIABLES PER SEGMENT	= = :	= = :
PARAMETER	AI < AT = DT : 0.1748	= = :
LOCAL	= = :	= = :

PARAMETER PASSAGE TYPE PERCENTAGES :	= = :	AI < AT = DT : 0.1606
VALUE	= = :	AI < AT = DT : 0.1606
REFERENCE	= = :	
(SEG,GLOBAL) ACTUAL USAGE PAIRS	= = :	= = :
ENTRY	= = :	= = :
MODIFIED	= = :	= = :
UNMODIFIED	= = :	= = :
NONENTRY	= = :	= = :
MODIFIED	= = :	= = :
UNMODIFIED	= = :	= = :
MODIFIED	= = :	AT < DT = AI : 0.1061
UNMODIFIED	= = :	= = :
(SEG,GLOBAL) POSSIBLE USAGE PAIRS	AI < AT = DT : 0.1227	AI < DT < AT : 0.0523
ENTRY	= = :	= = :
MODIFIED	= = :	= = :
UNMODIFIED	= = :	= = :
NONENTRY	= = :	AI < AT = DT : 0.0786
MODIFIED	= = :	DT = AI < AT : 0.0510
UNMODIFIED	= = :	AI < DT < AT : 0.1727
MODIFIED	= = :	= = :
UNMODIFIED	= = :	= = :
(SEG,GLOBAL) USAGE RELATIVE PERCENTAGES	= = :	= = :
ENTRY	AT < DT < AI : 0.1173	= = :
MODIFIED	AT < DT < AI : 0.1232	= = :
UNMODIFIED	= = :	= = :
NONENTRY	= = :	= = :
MODIFIED	= = :	= = :
UNMODIFIED	AT < DT = AI : 0.1546	= = :
MODIFIED	= = :	= = :
UNMODIFIED	= = :	= = :
(SEG,GLOBAL,SEG) DATA BINDINGS :	= = :	= = :
ACTUAL	= = :	= = :
SUBFUNCTIONAL	= = :	AI < AT = DT : 0.1963
INDEPENDENT	= = :	DT = AI < AT : 0.1529
POSSIBLE	DT = AI < AT : 0.1861	= = :
RELATIVE PERCENTAGE	= = :	= = :

\*\*\*\*\*

expected rejection percentage provides a comparative index of the true impact of the test results as a whole (in the example, a 25% actual rejection percentage would indicate that a truly significant effect, other than chance alone, was operative).

The point here may be illustrated in terms of simple coin-tossing experiments. The nature of statistics itself dictates that, out of a series of 100 separate statistical tests of a hypothetically fair coin at the .05 significance level, roughly 5 of those tests would nonetheless indicate that the coin was biased; if only 6 out of 100 tests of a real coin indicate bias at the .05 level, those six results have very little impact since the coin is behaving rather unbiasedly over the full set of tests.

This same "multiplicity" principle applies to the statistical conclusions of the study, since they represent the outcomes of a series of separate tests and were assumed in the statistical model to be separate experiments. It is appropriate to evaluate the location and dispersion results separately, since they reflect two separate issues (expectency and predictability) of software development behavior. Similarly, it is also appropriate to evaluate the process and product results separately. Finally, it is only fair to evaluate the "confirmatory" aspects as a distinct subset of all aspects examined, since they alone had been honestly considered prior to collecting and analyzing the data.

The details of this impact evaluation for the study's objective results, broken down into the appropriate categories identified above, are presented in the following table. The evaluation was performed at the  $\alpha=.20$  significance level used for screening purposes, hence the expected rejection percentage for any category was 20%. For each category of aspects, the table gives the number of (nonredundant) programming aspects, the expected (rounded to whole numbers) and actual numbers of rejections (of the null conclusion in favor of a directional alternative), and the expected and actual rejection percentages.

An asterisk marks those categories demonstrating noticeable statistical impact (i.e., actual rejection percentage well above expected rejection percentage).

category	number of aspects	expect. num. of reject.	actual num. of reject.	expect. reject. percent	actual reject. percent	
location	130	26	32	20.0	24.6	
process	10	2	9	20.0	90.0	*
confirmatory only	6	1	6	20.0	100.0	*
product	120	24	23	20.0	19.2	
confirmatory only	29	6	12	20.0	41.3	*
confirmatory only	35	7	18	20.0	51.4	*
dispersion	130	26	32	20.0	24.6	
process	10	2	2	20.0	20.0	
confirmatory only	6	1	0	20.0	0.0	
product	120	24	30	20.0	25.0	
confirmatory only	29	6	9	20.0	31.0	*
confirmatory only	35	7	9	20.0	25.7	

The table shows that the location results, dealing with the expectancy of software development behavior, do have statistical impact in several subcategories. Process aspects have more impact than product aspects on the whole, but when tempered by consideration of the distinction between "confirmatory" and "exploratory" aspects, the study's location results bear strong statistical impact for both process and product. They are better explained as the consequence of some true effect related to the experimental treatments, rather than as a random phenomenon.

It is also clear from the table that the dispersion results, dealing with the predictability of software development behavior, have little statistical impact in general. This is due primarily to the diminished power of statistical procedures used to test for dispersion differences, compounded by the small sample sizes involved and the coarseness of many of the programming aspects themselves. The lack of strong statistical impact in this area of the study does not mean that the dispersion issue is unimportant or undeserving of research attention, but rather that it is "a tougher nut to crack" than the location issue. The study's dispersion results are still worth pursuing, however, as possible hints of where differences might exist, provided this disclaimer regarding their impact is heeded.

### A Relaxed Differentiation View

As described in Section III, the research framework of possible three-way comparison outcomes provided the basis for converting the statistical results into the statistical conclusions. This framework has two inherent structural characteristics that may be exploited to make additional observations regarding the statistical conclusions. These structural characteristics and the supplemental views of the conclusions that they afford are described here and in the next subsection.

Specifically, the first structural characteristic is that each completely differentiated outcome is related to a specific pair of partially differentiated outcomes, as shown in the lattice of Diagram 2.1. For example,  $AI < AT < DT$ , a completely differentiated outcome, naturally weakens to either  $AI < AT = DT$  or  $AI = AT < DT$ , two partially differentiated outcomes.

Each completely differentiated outcome consists of three pairwise differences ( $AI < AT$ ,  $AT < DT$ ,  $AI < DT$  in the example), while each partially differentiated outcome consists of only two pairwise differences plus one pairwise equality ( $AI < DT$ ,  $AI < AT$ ,  $AT = DT$  and  $AI < DT$ ,  $AT < DT$ ,  $AI = AT$  in the example). The "outer" difference of the completely differentiated outcome ( $AI < DT$  in the example) is common to both partially differentiated outcomes, while each partially differentiated outcome focuses attention on one of the two "inner" differences ( $AI < AT$  and  $AT < DT$  in the example) to the exclusion of the other "inner" difference which is "relaxed" to an equality. Within a statistical environment or model which places a premium on claiming differences instead of equalities, a partially differentiated outcome is a safer statement, containing less error-prone information than a completely differentiated outcome. Since these outcomes represent statistical conclusions, the same data scores which support a completely differentiated outcome at a certain critical level also support each of the two related

partially differentiated outcomes at lower critical levels.

Thus, every completely differentiated conclusion may also be considered as two (more significant) partially differentiated conclusions, each of these three conclusions having equal and complete statistical legitimacy. The "outer" difference of a completely differentiated conclusion is, of course, stronger than either of its two "inner" differences; but the strengths of the two "inner" differences (relative to each other) will vary in accordance with the data scores and indeed are reflected in the significance levels of the two corresponding partially differentiated conclusions (relative to each other). Tables 4.1 and 4.2 give the details of this "relaxed differentiation" analysis for each of the completely differentiated conclusions found in the study, and an English paraphrase appears in Appendix 3. All of the partially differentiated conclusions listed in these tables should be added to those presented in Tables 2 and 3; they deserve full consideration in any analysis or interpretation of the study's findings. However, in the case that one of a partially differentiated pair is noticeably stronger than the other, it is fair to consider only the stronger one for the purpose of analysis or interpretation dealing primarily with partially differentiated outcomes, since the study is mainly concerned with the most pronounced difference afforded by each aspect's data scores.

#### A Directionless View

The second structural characteristic of the possible outcome framework is that the outcomes may be classified into another closely related set of directionless outcomes, as shown in the lattice of Diagram 2.2. For example,  $AI < AT = DT$  and  $AT = DT < AI$ , two directional partially differentiated outcomes, both correspond to  $AI \neq AT = DT$ , a nondirectional partially differentiated outcome. All six of the directional completely differentiated outcomes correspond to the single nondirectional completely differentiated outcome  $AI \neq AT \neq DT$ .

Table 4.1 Relaxed Differentiation for Location Comparisons

programming aspect	completely differentiated conclusion	partially differentiated conclusions
	comparison :critical outcome : level	comparison :critical outcome : level
PROGRAM CHANGES	DT < AI < AT : 0.1848	DT < AI = AT : 0.0037 DT = AI < AT : 0.1846
LINES	AI < DT < AT : 0.1194	DT = AI < AT : 0.0617 AI < AT = DT : 0.1132
(SEG,GLOBAL) USAGE RELATIVE PERCENTAGES \ ENTRY	AT < DT < AI : 0.1173	AT < DT = AI : 0.0826 AT = DT < AI : 0.1111
(SEG,GLOBAL) USAGE RELATIVE PERCENTAGES \ ENTRY \ MODIFIED	AT < DT < AI : 0.1232	AT < DT = AI : 0.1132 AT = DT < AI : 0.1132

Table 4.2 Relaxed Differentiation for Dispersion Comparisons

programming aspect	completely differentiated conclusion	partially differentiated conclusions
	comparison :critical outcome : level	comparison :critical outcome : level
MAX UNIQUE COMPILATIONS F.A.O. MODULE	DT < AI < AT : 0.0514	DT < AI = AT : 0.0036 DT = AI < AT : 0.0511
STATEMENT TYPE COUNTS \ RETURN	DT < AI < AT : 0.1398	DT = AI < AT : 0.0035 DT < AI = AT : 0.1395
(SEG,GLOBAL) POSSIBLE USAGE PAIRS	AI < DT < AT : 0.0523	AI < AT = DT : 0.0207 DT = AI < AT : 0.0511
(SEG,GLOBAL) POSSIBLE USAGE PAIRS \ NONENTRY \ UNMODIFIED	AI < DT < AT : 0.1727	AI < AT = DT : 0.1167 DT = AI < AT : 0.1561

By emphasizing only the observed distinctions between the groups, these directionless outcome categories focus attention on the original research issue of how observable programming aspects reflect differences among the three programming environments. In particular, there are three nondirectional partially differentiated outcomes (each of the form "one group different from the other two which are similar"), and it is noteworthy to observe just what set of programming aspects supports each of these basic distinctions. It is fairly easy to coalesce the directional distinctions from Table 2 into the directionless categories by eye, but a complete itemization of directionless distinctions is provided in Appendix 4. It is interesting to note that, for location comparisons, the directionless distinctions segregate cleanly along the process versus product dicotomy line: all of the product distinctions fall into the  $AI \neq AT = DT$  and  $AT \neq DT = AI$  directionless categories, while all of the process distinction fall into the  $DT \neq AI = AT$  directionless category.

### Individual Highlights

The purpose of this concluding subsection is simply to draw attention to what seem to be the "top ten" (or so) most noteworthy conclusions from among the study's objective results. These conclusions are interesting individually, either because the programming aspect itself has general appeal or because the difference in behavior expectancy or predictability is well pronounced (as indicated by a low critical significance level) in the experimental sample data.

Noteworthy location distinctions are mentioned below.

1. According to the  $DT < AI = AT$  outcome on the COMPUTER JOB STEPS aspect, the disciplined teams used very noticeably fewer computer job steps (i.e., module compilations, program executions, or miscellaneous job steps) than both the ad hoc individuals and the ad hoc teams.
2. This same difference was apparent in the total number of module compilations, the number of unique (i.e., not an



identical recompilation of a previously compiled module) module compilations, the number of program executions, and the number of essential job steps (i.e., unique module compilations plus program executions), according to the  $DT < AI = AT$  outcomes on the COMPUTER JOB STEPS\MODULE COMPILATIONS, COMPUTER JOB STEPS\MODULE COMPILATIONS\UNIQUE, COMPUTER JOB STEPS\PROGRAM EXECUTIONS, and ESSENTIAL JOB STEPS aspects, respectively.

3. According to the  $DT < AI = AT$  outcome on the PROGRAM CHANGES aspect, the disciplined teams required fewer textual revisions to build and debug the software than the ad hoc individuals and the ad hoc teams.
4. There was a definite trend for the ad hoc individuals to have produced fewer total symbolic lines (includes comments, compiler directives, statements, declarations, etc.) than the disciplined teams who produced fewer than the ad hoc teams, according to the  $AI < DT < AT$  outcome on the LINES aspect.
5. According to the  $AI < AT = DT$  outcome on the SEGMENTS aspect, the ad hoc individuals organized their software into noticeably fewer routines (i.e., functions or procedures) than either the ad hoc teams or the disciplined teams.
6. The ad hoc individuals displayed a trend toward having a greater number of statements per routine than did either the ad hoc teams or the disciplined teams, according to the  $AT = DT < AI$  outcome on the AVERAGE STATEMENTS PER SEGMENT aspect.
7. According to the  $DT = AI < AT$  outcomes on the STATEMENT TYPE COUNTS\IF and STATEMENT TYPE PERCENTAGE\IF aspects, both the ad hoc individuals and the disciplined teams coded noticeably fewer IF statements than the ad hoc teams, in terms of both total number and percentage of total statements.
8. According to the  $DT = AI < AT$  outcome on the DECISIONS aspect, both the ad hoc individuals and the disciplined teams tended to code fewer decisions (i.e., IF, WHILE, or CASE statements) than the ad hoc teams.
9. Both the ad hoc teams and the disciplined teams declared a noticeably larger number of data variables (i.e., scalars or

arrays of scalars) than the ad hoc individuals, according to the  $AI < AT = DT$  outcome on the DATA VARIABLES aspect.

10. According to the  $AT = DT < AI$  outcome on the DATA VARIABLE SCOPE PERCENTAGES\NONGLOBAL\LOCAL aspect, the ad hoc individuals had a larger percentage of local variables compared to the total number of declared data variables than either the ad hoc teams or the disciplined teams.
11. There was a slight trend for both the ad hoc individuals and the disciplined teams to have fewer potential data bindings [Stevens, Myers, and Constantine 74] (i.e., occurrences of the situation where a global variable could be modified by one segment and accessed by another due to the software's modularization) than the ad hoc teams, according to the  $DT = AI < AT$  outcome on the (SEG,GLOBAL,SEG) DATA BINDINGS\POSSIBLE aspect.

Noteworthy dispersion distinctions are mentioned below.

1. There was a noticeable difference in variability, with the disciplined teams less than the ad hoc individuals less than the ad hoc teams, in the maximum number of unique compilations for any one module, according to the  $DT < AI < AT$  outcome on the MAX UNIQUE COMPILATIONS F.A.O. MODULE aspect.
2. The ad hoc individuals exhibited noticeably greater variation than either the ad hoc teams or the disciplined teams in the number of miscellaneous job steps (i.e., auxiliary compilations or executions of something other than the final software project), according to the  $AT = DT < AI$  outcome on the COMPUTER JOB STEPS\MISCELLANEOUS aspect.
3. According to the  $DT = AI < AT$  outcome on the AVERAGE SEGMENTS PER MODULE aspect, the ad hoc individuals and the disciplined teams both exhibited noticeably less variation in the average number of routines per module than the ad hoc teams.
4. According to the  $DT = AI < AT$  outcomes on the STATEMENT TYPE COUNTS\RETURN and STATEMENT TYPE PERCENTAGES\RETURN aspects, the ad hoc teams showed rather noticeably greater variability in the number (both raw count and normalized percentage) of

RETURN statements coded than both the disciplined teams and the ad hoc individuals.

5. In the number of calls to programmer-defined routines, the ad hoc individuals displayed noticeably greater variation than both the ad hoc teams and the disciplined teams, according to the  $AT = DT < AI$  outcome on the `INVOCATIONS\NONINTRINSIC` aspect.
6. According to the  $DT < AI = AT$  outcome on the `DATA VARIABLES SCOPE PERCENTAGES\GLOBAL\NONENTRY\MODIFIED` aspect, the disciplined teams displayed noticeably smaller variation than either the ad hoc individuals or the ad hoc teams in the percentage of commonplace (i.e., ordinary scope and modified during execution) global variables compared to the total number of data variables declared.
7. The ad hoc individuals displayed noticeably less variation in the number of formal parameters passed by reference than both the ad hoc teams and the disciplined teams, according to the  $AI < AT = DT$  outcome on the `DATA VARIABLE SCOPE COUNTS\NONGLOBAL\PARAMETER\REFERENCE` aspect.
8. According to the  $AI < DT < AT$  outcome on the `(SEG,GLOBAL) POSSIBLE USAGE PAIRS` aspect, there was a noticeable difference in variability, with the ad hoc individuals less than the disciplined teams less than the ad hoc teams, for the total number of possible segment-global usage pairs (i.e., occurrences of the situation where a global variable could be modified or accessed by a segment).
9. According to the  $DT = AI < AT$  outcome on the `(SEG,GLOBAL,SEG) DATA BINDINGS\POSSIBLE` aspect, the ad hoc teams tended toward greater variability than either the ad hoc individuals or the disciplined teams in the number of potential data bindings.

## V. Interpretive Results

This section reports the interpretive results of the study, namely the research interpretations based on the conclusions presented in Section IV. The tone of discussion here is purposely somewhat subjective and opinionated, since the study's most important results are derived from interpreting the experiment's immediate findings in view of the study's overall goals. These interpretations also express the researchers' own estimation of the study's implications and general import according to their professional intuitions about programming and software.

The interpretations presented here are neither exhaustive nor unique. They only touch upon certain overall issues and generally avoid attaching meaning to or giving explanation for individual aspects or outcomes. It is anticipated that the reader and other researchers might formulate additional or alternative interpretations of the study's factual findings, using their own intuitive judgments.

Two distinct sets of research interpretations are discussed in the remainder of this section. The first set states general trends in the conclusions according to the basic suppositions of the study. The second set states general trends in the conclusions based on classifications which reflect certain abstract programming notions (e.g., cost, modularity, data organizations, etc.).

### According to Basic Suppositions:

The study's basic suppositions are a set of the simplest a priori expectations (or "hypotheses") for the outcomes of location and dispersion comparisons on process and product aspects. They are stated in the following table:

Basic Suppositions	on Location and Dispersion Comparisons
for Process Aspects	$DT < AI = AT$
for Product Aspects	$DT = AI < AT$ or $AT < DT = AI$

The basic suppositions are founded upon certain general beliefs regarding software development, which had been formulated by the researchers prior to conducting the experiment. The principal beliefs are that

- (a) methodological discipline is the key influence on the general efficiency of the process itself,
- (b) the disciplined methodology reduces the cost and complexity of the process and enhances the predictability of the process as well,
- (c) the preferred direction of both location and dispersion differences on process aspects is clear and undebatable, due to the tangibility of the process aspects themselves and the direct applicability of expected values and variations in terms of average cost estimates and tightness of cost estimates,
- (d) "mental cohesiveness" (or conceptual integrity [Brooks 75; pp. 41-50]) is the key influence on the general quality of the product itself,
- (e) a programming team is naturally burdened (relative to an individual programmer) by the organizational overhead and risk of error-prone misunderstanding inherent in coordinating and interfacing the thoughts and efforts of those on the team,
- (f) the disciplined methodology induces an effective mental cohesiveness, enabling a programming team to behave more like an individual programmer with respect to conceptual control over the program, its design, its structure, etc., because of the discipline's antiregressive, complexity-controlling [Belady and Lehman 76; p. 245] effect that compensates for the inherent organizational overhead of a team, and
- (g) the preferred direction of both location and dispersion

differences on product aspects is not always clear (occasionally subject to diverging viewpoints), due to the intangibility of many of the product aspects and a general lack of understanding regarding the implication of dispersion comparisons themselves for product aspects.

Against the background of these general beliefs and basic suppositions, each possible comparison outcome takes on a new meaning, depending on whether it would substantiate or contravene the general beliefs. For process aspects,

- (1) outcome  $DT < AI = AT$ , the supposition itself, is directly supportive of the beliefs;
- (2) outcomes  $DT < AI < AT$  and  $DT < AT < AI$ , which are completely differentiated variations of the supposition's main theme, are indirectly supportive of the beliefs, especially when  $DT < AI = AT$  is the stronger of the two corresponding partially differentiated outcomes;
- (3) outcome  $AI = AT = DT$  may discredit the beliefs, or it may be considered neutral for anyone of several possible reasons [(a) the critical level for a non-null outcome is just not low enough, so the aspect defaults to the null outcome; (b) the aspect simply reflects something characteristic of the application itself (or another factor common to all the groups in the experiment); or (c) the aspect actually measures something fundamental to the software development phenomenon in general and would always result in the null outcome]; and
- (4) all other outcomes discredit the beliefs.

For product aspects,

- (1) outcomes  $AT \neq DT = AI$  [ $AT < DT = AI$ ,  $DT = AI < AT$ ], the supposition itself, are directly supportive of the beliefs;
- (2) outcomes  $AI < DT < AT$  and  $AT < DT < AI$ , which may be considered as approximations of the suppositions (DT is distinct from AT but falls short of AI, due to lack of

- experience or maturity in the disciplined methodology), are indirectly supportive of the beliefs, especially when  $DT = AI < AT$  and  $AT < DT = AI$  (respectively) are the stronger of the two corresponding partially differentiated outcomes;
- (3) outcome  $AI = AT = DT$  may discredit the beliefs, or it may be considered neutral for anyone of several possible reasons [(a) the critical level for a non-null outcome is just not low enough, so the aspect defaults to the null outcome; (b) the aspect simply reflects something characteristic of the application itself (or another factor common to all the groups in the experiment); (c) the aspect actually measures something fundamental to the software development phenomenon in general and would always result in the null outcome; or (d) several of the study's hit-and-miss collection of "exploratory" product aspects are simply duds and may be ignored as useless software measures]; and
- (4) all other outcomes discredit one or more of the beliefs.

Thus the interpretation of the study's findings according to the basic suppositions consists simply of a general assessment of how well the research conclusions have borne out the basic suppositions and how well the experimental evidence substantiates the general beliefs. On the whole, the study's findings positively support the general beliefs presented above, although a few conclusions exist which are directly inconsistent with the suppositions or difficult to allay individually.

Support for the beliefs was relatively stronger on process aspects than on product aspects, and in location comparisons than in dispersion comparisons. Overwhelming support came in the category of location comparisons on process aspects in which the research conclusions are distinguished by extremely low critical levels and by near unanimity with the basic supposition. In the category of dispersion comparisons on process aspects, only two outcomes indicated any distinction among the groups: one aspect

supported the study's beliefs and one aspect showed an explainable exception to them. Fairly strong support also came in the category of location comparisons on product aspects for which the only negative evidence (besides the neutral  $AI = AT = DT$  conclusions) appeared in the form of several  $AI \neq AT = DT$  conclusions. They indicate some areas in which the disciplined methodology was apparently ineffective in modifying a team's behavior toward that of an individual, probably due to a lack of fully developed training/experience with the methodology. Comparatively weaker support for the study's beliefs was recorded in the category of dispersion comparisons on product aspects. Although the suppositions were borne out in a number of the conclusions, there were also several distinctions of various forms which contravene the suppositions.

Thus, according to this interpretation, the study's findings strongly substantiate the claims that

- (a) methodological discipline is the key influence on the general efficiency of the software development process, and that
- (b) the disciplined methodology significantly reduces the material costs of software development.

The claims that

- (a) mental cohesiveness is the key influence on the general quality of the software development product, that
  - (b) an ad hoc team is mentally burdened by organizational overhead compared to an individual, and that
  - (c) the disciplined methodology offsets the mental burden of organizational overhead and enables a team to behave more like an individual relative to the product itself,
- are moderately substantiated by the study's findings, with particularly mixed evidence for dispersion comparisons on product aspects.

It should be noted that there is a simpler, better-supported interpretive model for the location results alone. With the beliefs that a disciplined methodology provides for the minimum



process cost and results in a product which in some aspects approximates the product of an individual and at worst approximates the product developed by an ad hoc team, the suppositions are  $DT \leq AI, AT$  with respect to process and  $AI \leq DT \leq AT$  or  $AT \leq DT \leq AI$  with respect to product. The study's findings support these suppositions without exception.

According to Programming Aspect Classification:

Before presenting the interpretations according to a classification of the programming aspects, an explanation is in order regarding this classification and its motivation. It is desirable to make general interpretations in view of the way certain general programming issues are reflected among the individual programming aspects. For this purpose, the aspects considered in this study were grouped into (so-called) programming aspect classes. Each class consists of aspects which are related by some common feature (for example, all aspects relating to the program's statements, statement types, statement nesting, etc.), and the classes are not necessarily disjoint (i.e., a given aspect may be included in two or more classes). A unique higher-level programming issue (in the example, control structure organization) is associated with each class.

The programming aspects of this study were organized into a hierarchy of nine aspect classes (with about 10% overlap overall), outlined as follows:

<u>Higher-level Programming Issue:</u>	<u>Class:</u>
Development Process Efficiency	
Effort (Job Steps) . . . . .	I
Errors (Program Changes) . . . . .	II
Final Product Quality	
Gross Size . . . . .	III
Control-Construct Structure . . . . .	IV
Data Variable Organization . . . . .	V
Modularity	
Packaging Structure . . . . .	VI
Invocation Organization . . . . .	VII
Inter-Segment Communication	
Via Parameters . . . . .	VIII
Via Global Variables . . . . .	IX

The individual aspects comprising each class, together with the corresponding conclusions, are listed by classes in Tables 5.1

Table 5.1 Conclusions for Class I, Effort (Job Steps)

```

*****
|          programming aspect          |          location          |          dispersion          |
|          comparison :critical       |          comparison :critical |          comparison :critical |
|          outcome   : level         |          outcome   : level   |          outcome   : level   |
*****
| COMPUTER JOB STEPS                  | DT < AI = AT : 0.0036      | = = :                        |
|   MODULE COMPILATIONS              | DT < AI = AT : 0.0223      | = = :                        |
|     UNIQUE                          | DT < AI = AT : 0.0110      | = = :                        |
|     IDENTICAL                       | = = :                      | = = :                        |
|   PROGRAM EXECUTIONS               | DT < AI = AT : 0.0221      | = = :                        |
|   MISCELLANEOUS                    | DT < AI = AT : 0.1445      | AT = DT < AI : 0.0775      |
|-----|-----|-----|
| ESSENTIAL JOB STEPS                | DT < AI = AT : 0.0037      | = = :                        |
|-----|-----|-----|
| AVERAGE UNIQUE COMPILATIONS PER MODULE | DT < AI = AT : 0.0883      | = = :                        |
| MAX UNIQUE COMPILATIONS F.A.O. MODULE | DT < AI = AT : 0.1180      | DT < AI < AT :& .0514      |
*****
    
```

alternative conclusions (from Table 4) showing relaxed differentiation:  
 (correspondence indicated via the & symbol)

```

*****
|          |          | DT < AI = AT :& .0036 |
|          |          | DT = AI < AT :& .0511 |
*****
    
```

Table 5.2 Conclusions for Class II, Errors (Program Changes)

```

*****
|          programming aspect          |          location          |          dispersion          |
|          comparison :critical       |          comparison :critical |          comparison :critical |
|          outcome   : level         |          outcome   : level   |          outcome   : level   |
*****
| PROGRAM CHANGES                    | DT < AI < AT :& .1848      | = = :                        |
*****
    
```

alternative conclusions (from Table 4) showing relaxed differentiation:  
 (correspondence indicated via the & symbol)

```

*****
|          |          | DT < AI = AT :& .0037 |
|          |          | DT = AI < AT :& .1846 |
*****
    
```

Table 5.3 Conclusions for Class III, Gross Size

programming aspect	location	dispersion
	comparison :critical outcome : level	comparison :critical outcome : level
MODULES	= = :	= = :
AVERAGE SEGMENTS PER MODULE	= = :	DT = AI < AT : 0.0218
AVERAGE GLOBAL VARIABLES PER MODULE	= = :	= = :
SEGMENTS	AI < AT = DT : 0.0634	= = :
AVERAGE STATEMENTS PER SEGMENT	AT = DT < AI : 0.1706	= = :
AVERAGE NONGLOBAL VARIABLES PER SEGMENT	= = :	= = :
PARAMETER LOCAL	AI < AT = DT : 0.1748	= = :
	= = :	= = :
DATA VARIABLES	AI < AT = DT : 0.0698	= = :
DATA VARIABLE SCOPE COUNTS \ GLOBAL	AI < AT = DT : 0.1476	AI = AT < DT : 0.1241
DATA VARIABLE SCOPE COUNTS \ NONGLOBAL	= = :	= = :
PARAMETER LOCAL	AI < AT = DT : 0.1271	AI = AT < DT : 0.1061
	= = :	= = :
LINES	AI < DT < AT :& .1194	= = :
STATEMENTS	= = :	AT < DT = AI : 0.1954
AVERAGE TOKENS PER STATEMENT	= = :	AI = AT < DT : 0.1061
TOKENS	= = :	= = :

alternative conclusions (from Table 4) showing relaxed differentiation:  
(correspondence indicated via the & symbol)

DT = AI < AT :& .0617
AI < AT = DT :& .1132

Table 5.4 Conclusions for Class IV, Control-Construct Structure

programming aspect	location comparison :critical outcome : level	dispersion comparison :critical outcome : level
STATEMENTS	= = :	AT < DT = AI : 0.1954
STATEMENT TYPE COUNTS :	:	:
:=	= = :	= = :
IF	DT = AI < AT : 0.0780	= = :
CASE	= = :	= = :
WHILE	= = :	= = :
EXIT	= = :	= = :
(PROC) CALL	= = :	DT < AI = AT : 0.0325
NONINTRINSIC	= = :	DT < AI = AT : 0.1862
INTRINSIC	DT = AI < AT : 0.1732	= = :
RETURN	DT = AI < AT : 0.0860	DT < AI < AT :& .1398
STATEMENT TYPE PERCENTAGES :	:	:
:=	= = :	= = :
IF	DT = AI < AT : 0.1069	= = :
CASE	= = :	= = :
WHILE	= = :	= = :
EXIT	= = :	= = :
(PROC) CALL	= = :	= = :
NONINTRINSIC	= = :	= = :
INTRINSIC	= = :	= = :
RETURN	= = :	DT = AI < AT : 0.0401
AVERAGE STATEMENT NESTING LEVEL	= = :	= = :
DECISIONS	DT = AI < AT : 0.1468	= = :
FUNCTION CALLS	= = :	= = :
NONINTRINSIC	= = :	= = :
INTRINSIC	= = :	= = :

alternative conclusions (from Table 4) showing relaxed differentiation:  
(correspondence indicated via the & symbol)

	DT = AI < AT :& .0035
	DT < AI = AT :& .1395

Table 5.5 Conclusions for Class V, Data Variable Organization

programming aspect	location comparison :critical outcome : level	dispersion comparison :critical outcome : level
DATA VARIABLES	AI < AT = DT : 0.0698	= = :
DATA VARIABLE SCOPE COUNTS :		
GLOBAL	AI < AT = DT : 0.1476	AI = AT < DT : 0.1241
ENTRY	= = :	= = :
MODIFIED	= = :	= = :
UNMODIFIED	= = :	= = :
NONENTRY	= = :	= = :
MODIFIED	= = :	= = :
UNMODIFIED	= = :	= = :
MODIFIED	AI < AT = DT : 0.1614	= = :
UNMODIFIED	= = :	= = :
NONGLOBAL	= = :	= = :
PARAMETER	AI < AT = DT : 0.1271	AI = AT < DT : 0.1061
VALUE	= = :	= = :
REFERENCE	= = :	AI < AT = DT : 0.0199
LOCAL	= = :	= = :
DATA VARIABLE SCOPE PERCENTAGES :		
GLOBAL	= = :	AI = AT < DT : 0.0750
ENTRY	= = :	= = :
MODIFIED	= = :	= = :
UNMODIFIED	= = :	= = :
NONENTRY	= = :	= = :
MODIFIED	= = :	DT < AI = AT : 0.0218
UNMODIFIED	= = :	= = :
MODIFIED	= = :	= = :
UNMODIFIED	= = :	= = :
NONGLOBAL	= = :	AI = AT < DT : 0.0750
PARAMETER	AI < AT = DT : 0.1507	AI = AT < DT : 0.0557
VALUE	= = :	AI = AT < DT : 0.0943
REFERENCE	= = :	AI < AT = DT : 0.1529
LOCAL	AT = DT < AI : 0.1090	= = :
AVERAGE GLOBAL VARIABLES PER MODULE	= = :	= = :
ENTRY	= = :	= = :
NONENTRY	= = :	= = :
MODIFIED	= = :	DT = AI < AT : 0.1100
UNMODIFIED	= = :	= = :
AVERAGE NONGLOBAL VARIABLES PER SEGMENT	= = :	= = :
PARAMETER	AI < AT = DT : 0.1748	= = :
LOCAL	= = :	= = :

Table 5.6 Conclusions for Class VI, Packaging Structure

programming aspect	location	dispersion
	comparison :critical outcome : level	comparison :critical outcome : level
MODULES	= = :	= = :
AVERAGE SEGMENTS PER MODULE	= = :	DT = AI < AT : 0.0218
AVERAGE GLOBAL VARIABLES PER MODULE	= = :	= = :
SEGMENTS	AI < AT = DT : 0.0634	= = :
SEGMENT TYPE COUNTS \ FUNCTION	= = :	= = :
SEGMENT TYPE COUNTS \ PROCEDURE	= = :	= = :
SEGMENT TYPE PERCENTAGES \ FUNCTION	= = :	= = :
SEGMENT TYPE PERCENTAGES \ PROCEDURE	= = :	= = :
AVERAGE STATEMENTS PER SEGMENT	AT = DT < AI : 0.1706	= = :
AVERAGE NONGLOBAL VARIABLES PER SEGMENT	= = :	= = :
PARAMETER	AI < AT = DT : 0.1748	= = :
LOCAL	= = :	= = :

Table 5.7 Conclusions for Class VII, Invocation Organization

programming aspect	location	dispersion
	comparison :critical outcome : level	comparison :critical outcome : level
INVOCATIONS	= = :	AT = DT < AI : 0.0206
FUNCTION	= = :	= = :
NONINTRINSIC	= = :	= = :
INTRINSIC	= = :	= = :
PROCEDURE	= = :	DT < AI = AT : 0.0325
NONINTRINSIC	= = :	DT < AI = AT : 0.1862
INTRINSIC	DT = AI < AT : 0.1732	= = :
NONINTRINSIC	= = :	AT = DT < AI : 0.0510
INTRINSIC	DT = AI < AT : 0.0435	= = :
AVG INVOCATIONS PER (CALLING) SEGMENT	= = :	= = :
FUNCTION	= = :	= = :
NONINTRINSIC	= = :	= = :
INTRINSIC	= = :	= = :
PROCEDURE	= = :	= = :
NONINTRINSIC	= = :	= = :
INTRINSIC	= = :	DT < AI = AT : 0.0653
NONINTRINSIC	AT = DT < AI : 0.1699	= = :
INTRINSIC	= = :	= = :
AVG INVOCATIONS PER (CALLED) SEGMENT	AT = DT < AI : 0.1699	= = :
FUNCTION	AT = DT < AI : 0.1936	AT < DT = AI : 0.1411
PROCEDURE	= = :	= = :

Table 5.8 Conclusions for Class VIII, Communication via Parameters

programming aspect	location	dispersion
	comparison :critical outcome : level	comparison :critical outcome : level
DATA VARIABLE SCOPE COUNTS\NONGLOBAL :	:	:
PARAMETER	AI < AT = DT : 0.1271	AI = AT < DT : 0.1061
VALUE	= = :	= = :
REFERENCE	= = :	AI < AT = DT : 0.0199
AVG NONGLOBAL VARIABLES PER SEGMENT :	= = :	= = :
PARAMETER	AI < AT = DT : 0.1748	= = :
PARAMETER PASSAGE TYPE PERCENTAGES :	:	:
VALUE	= = :	AI < AT = DT : 0.1606
REFERENCE	= = :	AI < AT = DT : 0.1606

Table 5.9 Conclusions for Class IX, Communication via Global Variables

programming aspect	location comparison :critical outcome : level	dispersion comparison :critical outcome : level
DATA VARIABLE SCOPE COUNTS \ GLOBAL	AI < AT = DT : 0.1476	AI = AT < DT : 0.1241
ENTRY	= = :	= = :
MODIFIED	= = :	= = :
UNMODIFIED	= = :	= = :
NONENTRY	= = :	= = :
MODIFIED	= = :	= = :
UNMODIFIED	= = :	= = :
MODIFIED	AI < AT = DT : 0.1614	= = :
UNMODIFIED	= = :	= = :
AVERAGE GLOBAL VARIABLES PER MODULE	= = :	= = :
ENTRY	= = :	= = :
NONENTRY	= = :	= = :
MODIFIED	= = :	DT = AI < AT : 0.1100
UNMODIFIED	= = :	= = :
(SEG,GLOBAL) ACTUAL USAGE PAIRS	= = :	= = :
ENTRY	= = :	= = :
MODIFIED	= = :	= = :
UNMODIFIED	= = :	= = :
NONENTRY	= = :	= = :
MODIFIED	= = :	= = :
UNMODIFIED	= = :	= = :
MODIFIED	= = :	AT < DT = AI : 0.1061
UNMODIFIED	= = :	= = :
(SEG,GLOBAL) POSSIBLE USAGE PAIRS	AI < AT = DT : 0.1227	AI < DT < AT : & .0523
ENTRY	= = :	= = :
MODIFIED	= = :	= = :
UNMODIFIED	= = :	= = :
NONENTRY	= = :	AI < AT = DT : 0.0786
MODIFIED	= = :	DT = AI < AT : 0.0510
UNMODIFIED	= = :	AI < DT < AT : @ .1727
MODIFIED	= = :	= = :
UNMODIFIED	= = :	= = :
(SEG,GLOBAL) USAGE RELATIVE PERCENTAGES	= = :	= = :
ENTRY	AT < DT < AI : # .1173	= = :
MODIFIED	AT < DT < AI : \$ .1232	= = :
UNMODIFIED	= = :	= = :
NONENTRY	= = :	= = :
MODIFIED	= = :	= = :
UNMODIFIED	AT < DT = AI : 0.1546	= = :
MODIFIED	= = :	= = :
UNMODIFIED	= = :	= = :
(SEG,GLOBAL,SEG) DATA BINDINGS :	= = :	= = :
ACTUAL	= = :	= = :
SUBFUNCTIONAL	= = :	= = :
INDEPENDENT	= = :	AI < AT = DT : 0.1963
POSSIBLE	DT = AI < AT : 0.1861	DT = AI < AT : 0.1529
RELATIVE PERCENTAGE	= = :	= = :

alternative conclusions (from Table 4) showing relaxed differentiation:  
(correspondence indicated via the &, @, #, and \$ symbols)

	AI < AT = DT : & .0207
	DT = AI < AT : & .0511
AT < DT = AI : # .0826	
AT = DT < AI : # .1111	
	AI < AT = DT : @ .1167
	DT = AI < AT : @ .1561
AT < DT = AI : \$ .1132	
AT = DT < AI : \$ .1132	