D3

INVESTIGATING SOFTWARE DEVELOPMENT APPROACHES: A SYNOPSIS *

Robert W. Reiter, Jr.
Department of Computer Science
University of Maryland
College Park, Maryland 20742

INTRODUCTION

The 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

The paper reports on research comparing various approaches, or methodologies, for software development. The study focuses ontthe 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 which used a disciplined methodology (consisting of top-down design, process design language usage, structured programming, chief programmer teams, and code reading) with programming teams and individual programmers which employed their own ad hoc approach. 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.

The purpose of the research was to develop an investigative methodology for experimentally studying and quantitatively characterizing the effect of methodologies and programming environments on software development. It involves the quantitative measurement and analysis of both the process and the product of software development, in manner which is minimally obstrusive (to those developing the software), very objective, and highly automatable. The basic premise is that distinctions among the groups exist both in the process and in the product.

SPECIFICS

Nineteen units (teams or individuals) each performed the same software development task, but under controlled and slightly varied conditions. Two programming factors, size of programming team and degree of methodological discipline, each with two levels (single individual, and three-person team; the ad hoc approach, and the disciplined methodology), were chosen as the independent variables and formed the experimental treatments. The dependent variables to be observed and measured were a large set (over 125) of programming aspects. The teams and individuals were

*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.

placed into three treatment groups, designated A, B and C (of 6, 6 and 7 units, respectively), each operating under a certain combination of factor-levels:

A – individuals, ad hoc approach;

B – three-person teams, ad hoc approach;

C – three-person teams, disciplined methodology.

The time and place for the experiment was Spring, 1976, in conjunction with two academic courses at the University of Maryland. The particular project or application to be developed was compiler for a small high-level language and a simple stack machine. This task was roughly a two man-month effort, and the resulting software systems averaged about 1200 source lines or 600 executable statements, in high-level structured-language code. The participants were advanced undergraduates and graduate students in the Computer Science Department. The implementation language was the high-level structured-programming language SIMPL-T [Basili and Turner 76], which is used extensivly in course work at the University and has string-processing capabilities similar to PL/1.

Data collection for the experiment was automated on-line, with essentially no interference to the programmer's normal pattern of actions during computer sessions. Special module compilation and program execution processors created an historical data base of source code and test data accumulated throughout the project development. Scores corresponding to each of the programming aspects were extracted directly and algorithmically from this data base.

The programming aspects represent specific automatically isolatable and observable features of the programming phenomenon, related to either the product or the process of software development. Product aspects are based on the syntactic content and organization of the symbolic source code which represents the complete final product developed. 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. Major headings for the particular programming aspects reported on in this study are listed in the accompanying table, with qualifying subcategories mentioned in square brackets.

APPROACH

The investigative methodology was designed and developed as a scientific and empirical solution to the problem of comparing software development efforts under various conditions. It was used to guide the planning, execution, and analysis of the set of experiments which comprise this study. The approach consists of eleven steps or elements, as shown in the accompanying schmatic diagram which charts the general flow (solid lines) and some of the interrelationships (dashed lines) among these elements.

The methodology begins with Questions of Interest, which are turned into Research Hypotheses and Statistical Hypotheses. The Statistical Model is very important since it governs the Experimental Design and several other elements. Statistical Results, corresponding directly to the Statistical Hypotheses, are determined by the Colledted Data via the Statistical Test Procedures. Research Framework(s) are necessary to organize the large volume of hypotheses and results into a smaller, more managable form as Statistical Conclusions and Research Interpretations.

RESULTS

The methodology provides that the study's results be separated into statistical conclusions, representing factual findings, and research interpretations, representing intuitive judgements.

For each aspect there is one statistical conclusion which states any differences observed among the three programming environments represented by the groups A, B, and C. These outcomes are expressed in the form of "equations"; e.g., A<B=C means that the average score for the individual programmers was appreciably lower than the average scores for the ad hoc teams and the disciplined teams which both had about the same average score. In addition to the null outcome (A=B=C) of no observed differences, there were twelve other possible outcomes, as noted in the accompanying table. The table simply lists all the non-null conclusions, arranged by outcome. The values in the "error" column state the risk, as a probability value, of erroneously making that conclusion and indicate how strongly pronounced the differences were in the data. Although there is much fascinating material in these findings, space permits only a few particularly interesting conclusions to be pointed out.

The A<B=C outcome was quite pronounced for the SEGMENTS aspect, indicating that the individuals built their systems with fewer routines on the average than either the ad hoc teams or the disciplined teams, which used about the same number of routines. According to the A<B=C and B=C<A outcomes, the individuals had noticably less global variables and more local variables than both types of teams. The C=A<B outcomes for IF statements and DECISIONS indicate aspects where the disciplined teams behaved like the individuals and both were different than the ad hoc teams. For the number of COMPUTER RUNS (JOB STEPS), and several subcategories, the C < A =B outcomes have very low error risks and indicate that the disciplined teams out-performed both the individuals and the ad hoc teams in these aspects. On the number of PROGRAM CHANGES – a measure of the amount of cummulative textual revision of the program source code during development, which has been shown to correlate well with total error occurrences [Dunsmore and Gannon 77] – the same data scores which support the C<A<B conclusion at a high risk of error (0.185) also support the C < A = B conclusion at a very low risk of error (0.004), indicating a strong distinction in terms of error-prone-ness in favor of the disciplined teams.

One framework for the interpretation of these conclusions is the concept of how the disciplined methodology actually impacts the software development process and product. Prior to conducting the experiment, certain general beliefs (see details on accompanying slide) about the impact had

Robert W. Reiter, Jr.
University of Maryland

been formulated. Certain basic suppositions (a priori expectations), for how the experiments should turn out if the beliefs were true, were constructed from the general beliefs. Examination of how the conclusions stack up against the suppositions (how true the beliefs are) shows that none of the conclusions for any of the observed programming aspects contravene the basic suppositions. Thus, the study's results may be interpreted as strong experimental evidence in favor of these general beliefs.

SUMMARY

A practical methodology was designed and developed for experimentally and quantitatively investigating the software development phenomenon. It was employed to compare three particular software development environments and to evaluate the relative impact of a particular disciplined methodology (made up of so-called modern programming practices). The experiments were successful in measuring differences among programming environments and the results support the claim that disciplined methodology effectively improves both the process and product of software development. The results will be used to guide further experiments and will act as a basis for analysis of software development products and processes in the Software Engineering Laboratory at NASA/GSFC [Easili et al. 77]. The intention is to persue this type of research, especially extending the study to include more sophisticated and promising programming aspects, such as Halstead's software science quantities [Halstead 77] and other software complexity metrics [McCabe 76].

REFERENCES

1. [Basili and Reiter 78] V.R. Basili and R.W. Reiter, Jr. Investigating Software Development Approaches. Technical Report TR-688, Department of Computer Science, University of Maryland, August, 1978.

2. [Basili and Turner 76] V.R. Basili and A.J. Turner. SIMPL-T, A Structured Programming Language. Paladin House Publishers, Geneva, Illinois. 1976.

3. [Basili et al. 77] V.R. Basili, M.V. Zelkowitz, F.E. McGarry, R.W. Reiter, Jr., W.F. Truszkowski, and D.L. Weiss. The Software Engineering Laboratory. Technical Report TR-535, Department of Computer Science, University of Maryland. May, 1977.

4. [Dunsmore and Gannon 77] P.E. Dunsmore and J.D. Gannon. Experimental Investigation of Programming Complexity. Proceedings of ACM-NES Sixteenth Annual Technical Symposium: Systems and Software (June 1977), Washington, D.C., pp. 117-125.

5. [Halstead 77] M. Halstead. Elements of Software Science. Elsevier Computer Science Library. 1977.

6. [McCabe 76] T.J. McCabe. A Complexity Measure. IEEE Transactions on Software Engineering, Vol. 2, No. 4 (December 1976), pp. 308-320.

Programming Aspects

****

Development Process Aspects :

COMPUTER RUNS (JOB STEPS)
[compilations, executions, miscellaneous]
ESSENTIAL RUNS (JOB STEPS)
AVERAGE UNIQUE COMPILATIONS PER MODULE
MAX UNIQUE COMPILATIONS FOR ANY ONE MODULE
PROGRAM CHANGES

****

Final Product Aspects :

MODULES
SEGMENTS
SEGMENT TYPE COUNTS
[function, procedure]
SEGMENT TYPE PERCENTAGES
[function, procedure]
AVERAGE SEGMENTS PER MODULE

LINES

STATEMENTS
STATEMENT TYPE COUNTS
[:=, IF, CASE,WHILE, EXIT, (proc)CALL, RETURN]
STATEMENT TYPE PERCENTAGES
[:=, IF, .CASE, WHILE, EXIT, (proc)CALL, RETURN]
AVERAGE STATEMENTS PER SEGMENT
AVERAGE STATEMENT NESTING LEVEL

DECISIONS

FUNCTION CALLS
[non-intrinsic, intrinsic]

TOKENS
AVERAGE TOKENS PER STATEMENT

INVOCATIONS
[function, procedure; non-intrinsic, intrinsic]
AVG INVOCATIONS PER (CALLING) SEGMENT
[function, procedure; non-intrinsic, intrinsic]
AVG INVOCATIONS PER (CALLED) SEGMENT
[function, procedure]

DATA VARIABLES
DATA VARIABLE SCOPE COUNTS
[global, parameter,local]
DATA VARIABLE SCOPE PERCENTAGES
[global, parameter, local]
AVERAGE GLOBAL VARIABLES PER MODULE
[modified, not modified; non-entry, entry]
AVERAGE NON-GLOBAL VARIABLES PER SEGMENT
[parameter, local]

PARAMETER PASSAGE TYPE PERCENTAGES
[value, reference]

(SEG,GLOBAL) ACTUAL USAGE PAIRS
[modified, not modified; non-entry, entry]
(SEG,GLOBAL) POSSIBLE USAGE PAIRS
[modified, not modified; non-entry, entry]
(SEG,GLOBAL) USAGE RELATIVE PERCENTAGES
[modified, not modified; non-entry, entry]

(SEG,GLOBAL,SEG) DATA BINDINGS
[actual; possible; relative percentage]

****

Robert W. Reiter, Jr.
University of Maryland

Robert W. Reiter, Jr.
University of Maryland

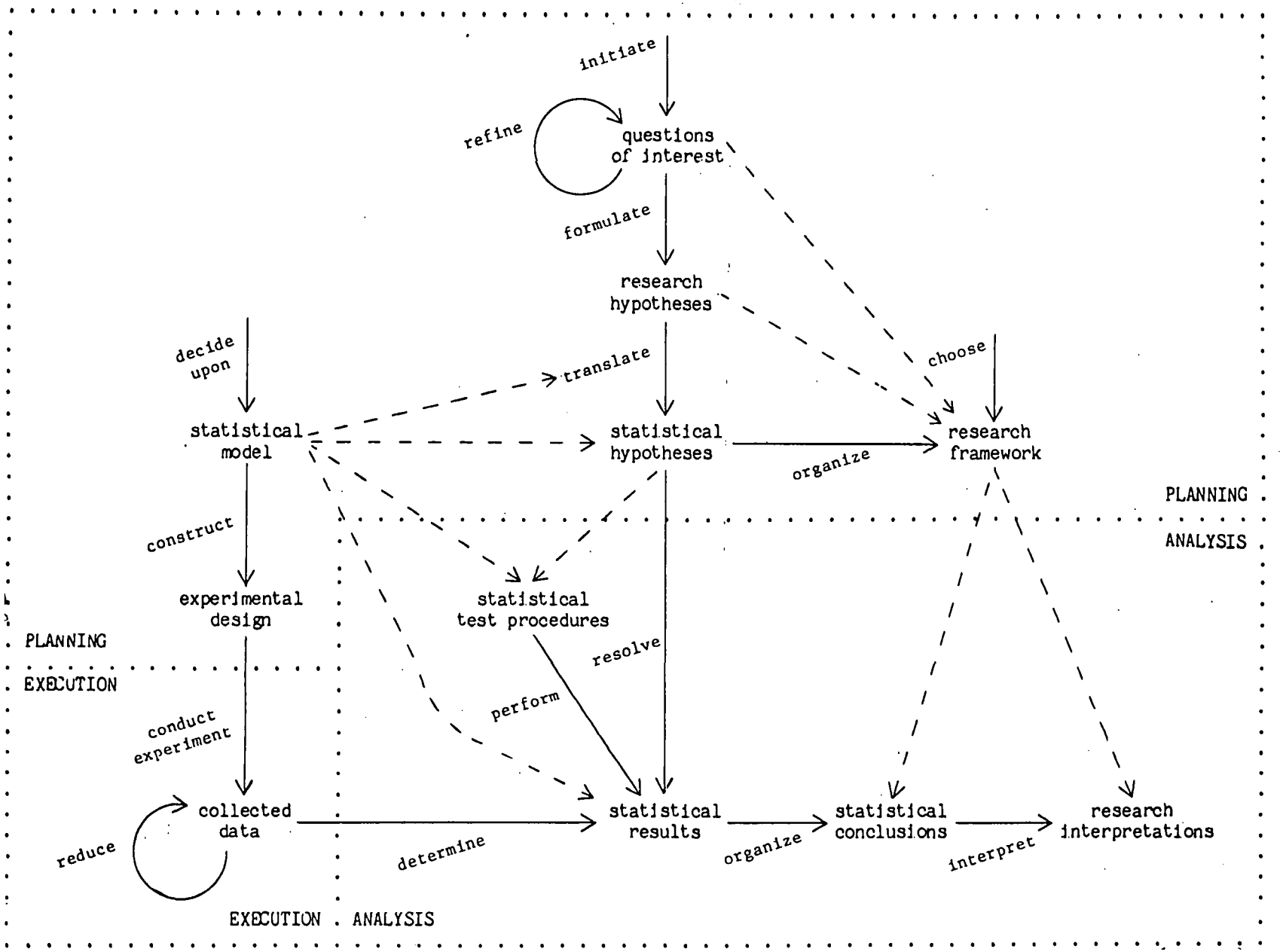

Diagram 1. Approach Schematic

Non-Null Conclusions, arranged by outcome

outcome	error freq	programming aspect
A < B = C	9	
	0.0634	SEGMENTS
	0.0698	DATA VARIABLES
	0.1476	DATA VARIABLE SCOPE COUNTS \ GLOBAL
	0.1614	DATA VARIABLE SCOPE COUNTS \ GLOBAL \ MODIFIED
	0.2015	DATA VARIABLE SCOPE COUNTS \ NON-GLOBAL
	0.1271	DATA VARIABLE SCOPE COUNTS \ NON-GLOBAL \ PARAMETER
	0.1507	DATA VARIABLE SCOPE PERCENTAGES \ NON-GLOBAL \ PARAMETER
	0.1748	AVERAGE NON-GLOBAL VARIABLES PER SEGMENT \ PARAMETER
	0.1227	(SEG,GLOBAL) POSSIBLE USAGE PAIRS
B = C < A	5	
	0.1706	AVERAGE STATEMENTS PER SEGMENT
	0.1699	AVG INVOCATIONS PER (CALLING) SEGMENT \ NON-INTRINSIC
	0.1699	AVG INVOCATIONS PER (CALLED) SEGMENT
	0.1936	AVG INVOCATIONS PER (CALLED) SEGMENT \ FUNCTION
	0.1090	DATA VARIABLE SCOPE PERCENTAGES \ NON-GLOBAL \ LOCAL
B < C = A	3	
	0.2195	STATEMENT TYPE PERCENTAGES \ CASE
	0.2364	(SEG,GLOBAL) USAGE RELATIVE PERCENTAGES
	0.1546	(SEG,GLOBAL) USAGE RELATIVE PERCENTAGES \ NOT MODIFIED \ NON-ENTRY
C = A < B	11	
	0.2134	SEGMENT TYPE COUNTS \ FUNCTION
	0.2321	STATEMENTS
	0.0780	STATEMENT TYPE COUNTS \ IF
	0.1732	STATEMENT TYPE COUNTS \ (PROC)CALL \ INTRINSIC
	0.0196	STATEMENT TYPE COUNTS \ RETURN
	0.1038	STATEMENT TYPE PERCENTAGES \ IF
	0.2065	STATEMENT TYPE PERCENTAGES \ RETURN
	0.1468	DECISIONS
	0.1732	INVOCATIONS \ PROCEDURE \ INTRINSIC
	0.0435	INVOCATIONS \ INTRINSIC
	0.1861	(SEG,GLOBAL,SEG) DATA BINDINGS \ POSSIBLE
C < A = B	8	
	0.0036	COMPUTER RUNS (JOB STEPS)
	0.0223	COMPUTER RUNS (JOB STEPS) \ MODULE COMPILATIONS
	0.0110	COMPUTER RUNS (JOB STEPS) \ MODULE COMPILATIONS \ UNIQUE
	0.0221	COMPUTER RUNS (JOB STEPS) \ PROGRAM EXECUTIONS
	0.1445	COMPUTER RUNS (JOB STEPS) \ MISCELLANEOUS
	0.0037	ESSENTIAL RUNS (JOB STEPS)
	0.0883	AVERAGE UNIQUE COMPILATIONS PER MODULE
	0.1180	MAX UNIQUE COMPILATIONS FOR ANY ONE MODULE
A = B < C	0	
A < B < C	0	
A < C < B	1	
	0.1194	LINES
B < C < A	2	
	0.1232	(SEG,GLOBAL) USAGE RELATIVE PERCENTAGES \ MODIFIED \ ENTRY
	0.1173	(SEG,GLOBAL) USAGE RELATIVE PERCENTAGES \ ENTRY
B < A < C	0	
C < A < B	1	
	0.1848	PROGRAM CHANGES
C < B < A	0	

Robert W. Reiter, Jr.
University of Maryland

Research Interpretations

General Beliefs:

- The disciplined methodology reduces the average cost and complexity of the process.
- The disciplined methodology can enable a programming team to compensate for their inherent coordination overhead and behave more like an individual programmer in terms of designing and building the product.

Basic Suppositions:

- on process aspects: $C \leqslant A, B$
- on product aspects: $A \leqslant C \leqslant B$ or $B \leqslant C \leqslant A$

Support from the conclusions:

- process:
  - $C < A = B$ on 8 aspects
  - $C < A < B$ on 1 aspect
  - $A = B = C$ on 1 aspect
- product:
  - $A < B = C$ on 9 aspects
  - $A = C < A$ on 5 aspects
  - $B < C = A$ on 3 aspects
  - $C = A < B$ on 11 aspects
  - $A < C < B$ on 1 aspect
  - $B < C < A$ on 2 aspects
  - $A = B = C$ on 96 aspects

None of the conclusions for any of the observed programming aspects contravene these basic suppositions.

Thus, the study's results may be interpreted as strong experimental evidence in favor of these general beliefs.

Robert W. Reiter, Jr.
University of Maryland