NBSIR 84-2827

Comparing Software Development Methodologies for Ada*: A Study Plan

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
Institute for Computer Sciences and Technology
Center for Programming Sciences and Technology
Washington, DC 20234

February 1984

*Ada is a trademark of the U.S. Department of Defense (AJPO)

U.S. DEPARTMENT OF COMMERCE
ONAL BUREAU OF STANDARDS

NBSIR 84-2827

COMPARING SOFTWARE DEVELOPMENT METHODOLOGIES FOR ADA*: A STUDY PLAN

Peter Freeman

Information and Computer Science
University of California, Irvine
Irvine, CA 92717

Anthony I. Wasserman

Medical Information Science
University of California, San Francisco
San Francisco, CA 94143

Edited by

Raymond C. Houghton, Jr.

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
Institute for Computer Sciences and Technology
Center for Programming Sciences and Technology
Washington, DC 20234

February 1984

*Ada is a trademark of the U.S. Department of Defense (AJPO)

U.S. DEPARTMENT OF COMMERCE, Malcolm Baldrige, *Secretary*
NATIONAL BUREAU OF STANDARDS, Ernest Ambler, *Director*

TABLE OF CONTENTS

PREFACE

The study that is outlined in this paper has been proposed as one of the early activities in the Support Systems Task Area of the DoD STARS Program (See IEEE Computer, November, 1983). This study is an update of the study that was originally proposed by Freeman and Wasserman in a group of papers titled, "Software Development Methodologies and Ada". These papers were published in a report submitted to the Ada Joint Program Office (AJPO) in November, 1982. The original study emphasized the design phase of a methodology. In particular, it attempted to isolate the design phase from the implementation phase so that one could study the effects of one phase on the other, e.g., a poor design followed by corrective implementation. In subsequent review, the editor and other reviewers felt that this level of differentiation added much initial complexity and risk to the study. Consequently, the version that follows treats the methodology as a "black box" and is therefore simpler than the original. After this study has been successfully completed, a future study should be planned based on the outcome of this one that attempts a higher degree of differentiation.

R. Houghton, Editor

NOTE: The mention of trade names for several software development methodologies does not constitute endorsement of those methodologies by the National Bureau of Standards.

Editing was performed under DoD contract AJPO-83-27.

MOTIVATION AND PROBLEM FORMULATION

Much has been said about the cost of software maintenance in the Department of Defense (DoD). In particular, one of the main goals of the Software Technology for Adaptable, Reliable Systems (STARS) program is to reduce the cost of maintenance. The motivation for the study described in this paper is to help determine the characteristics of software development methodologies that contribute to reduced costs for maintenance. When defined, these characteristics will form a basis for the definition of standards for development methodlogies.

Many recommendations have been made for improving the process of software development over the past 30 years. High level languages, timesharing systems, and programming techniques are among the ideas that have been successfully used. Many other ideas have been proposed, but not used extensively, making it difficult to determine their effectiveness.

The discipline of software engineering has emerged over the past 15 years as a focal point for efforts to improve both the quality of software products and the process by which they are created and evolved. After initial attempts in which isolated methods were developed to address different phases of the software lifecycle, recent work has concentrated on integration of methods.

The integration of technical methods and management procedures across the software lifecycle yields a methodology for software development - a process that can be systematically followed from the initial system concept through product release and operation. The methodology may be supported by automated tools, which may also be integrated into a programming support environment. Finally, the methodology and the programming support environment normally are used in a work setting, one or more physical locations in which the software development occurs.

While one would like to evaluate software development methodologies objectively, it is very difficult to do so since they are dependent upon the programming support environment, the software development organization applying the methodologies, and the physical setting of the organization. As a result, there are many variables to control, e.g. (1) the methodology itself, (2) the skill of the developers in the methodology, (3) the quality of the software tools in the support environment, (4) the typing speed of the developers, and (5) the degree to which the workspace permits uninterrupted concentration on the task at hand.

Thus, it is not possible to construct precise experiments involving methodologies in the same manner as one might perform experiments in psychology, using control groups and a limited number of independent variables. Nor is it possible to conduct experiments similar to those in the natural sciences, since it is difficult to obtain meaningful, statistically significant,

numerical data that characterize well what can be observed in this type of situation. However, it is possible to obtain useful information from carefully designed investigations of software maintenance.

In this context, the study has been formulated with the following objective:

> Determine what is the effect of various software development methodologies on the maintainability of systems, specifically ones that are constructed in Ada.

In short, we want to know which methods help to produce the most maintainable systems.

Furthermore, we wish to focus still further on the critical components of a methodology. For example, much attention has been paid to the detailed design aspects of software development (relative to efforts expended in other areas), and it is extremely important to understand which methods help most in establishing good software structure. The importance of structure (at the module organization level) is well understood by those who have looked deeply at the problem of software maintenance [e.g., L.A. Belady and M.M. Lehman, "A Model of Large Program Development," IBM Systems Journal 15,3 (1976), pp. 225-252]. What is less well understood is the effectiveness of different methods (or classes of methods) in helping software developers achieve a "good" structuring of a system (measured in terms of how easy it is to maintain the resulting system). Thus, a refined version of the key objective is:

> Determine how well various software development methodologies help structure systems built in Ada, as measured by the ease of maintenance of the resulting system.

There are, of course, many aspects of software development methodologies for which additional objectives can be defined, just as there are other aspects to investigate (for example, the correctness of the resulting systems). The choices are motivated by current perceptions of the relative importance of the various aspects of software development methodologies and the overall objectives of the Ada program. Likewise, in the hope of making a clear advance in the understanding of methods, the focus of this study is on software development rather than the larger sphere of system development. If this and other studies shed light on the software development problem, then extension of these results to the systems domain can be considered.

DESCRIPTIVE OVERVIEW

Software for the DoD is typically developed under contract. After development, it is turned over to a DoD agency for use. If maintenance is required on the software, it is either performed in-house, or it is contracted out to either the original developer or to a different (possibly low-bidding) contractor. In either case, it is rare that the individuals who developed the software are the same to also maintain it.

The cost of performing maintenance is dependent on how easy it is for a person to modify the software. Ease of modification depends on many factors, most of which relate back to the development methodology that was used. In particular, these characteristics are discussed in "Ada Methodologies: Concepts and Requirements" [Peter Freeman and Anthony I. Wasserman, "Software Development Methodologies and Ada", Software Engineering Notes, Vol. 8, No. 1, 1983]. These characteristics include understandability, readability, complexity, and correctness.

The comparative study described in the following paragraphs is modeled after the typical DoD software procurement. Figure 1 illustrates the overall structure and flow of information of the comparative study.

The study concentrates on one primary issue: the impact of alternative development methodologies on the maintainability of Ada code. To the extent possible, all parameters of the study have been chosen to maximize the collection of objective information on this issue. The basic elements of the study include:

1. experts in each of several methods (see Appendix 1) create Ada implementations for a specific problem (see Appendix 2);

2. each implementation is modified by each of several maintenance teams;

3. the impact of the methodology on the maintainability of the resulting Ada-coded systems is evaluated and reported.

Throughout the plan we have made assumptions and set parameters to control the variability of the investigation. To the extent that we have succeeded, the evaluation results will provide new insight into the relative ability of various characteristics of methodologies to reduce the cost of system evolution.

The investigation will be managed by an organization experienced in project management and DoD procedures. This prime contractor will run the investigation "development" as though it were a normal contract; greater than normal oversight will be needed, however, to perform data collection. A separate

contractor will perform an informal independent verification and validation (IV&V) function to insure that each "deliverable" (design, code, documentation, etc.) meets the substantive requirements of the project.

A single problem (outlined in Appendix 2) has been chosen that is representative of a fairly broad sample of the problems for which Ada is intended. We are not concerned with how easy it is to learn or use different methodologies, so established experts will be used to produce implementations that are as nearly perfect an application of each methodology as possible. Wherever feasible, the creator of a methodology should develop the implementation.

In order to more strongly reflect the DoD software acquisition environment, a change in requirements will be issued during the development phase. This change is also expected to discourage the developers from jumping too far ahead of schedule.

Variability in maintenance performance (caused by the learning effect of multiple maintenance tasks by a single team and differences in ability) will be controlled by using multiple maintenance teams. To further control variability, each maintenance team will maintain the systems in a different order, providing a basis for statistical analysis of the resulting data.

Evolution of each system will be carried out using the documentation that is developed as a product of the methodology, a maintenance manual (see Appendix 4), and specific maintenance instructions. The maintenance teams will carefully measure the effort required to make modifications. Secondary information will be obtained from the subjective evaluations of all parties involved in the study (see Appendix 5).

This structural overview of the study defines the general outlines of the investigation. The next section provides detailed explanation of each aspect, a detailed project plan, and some justifications for the particular choice of investigation parameters made. More extensive discussion of the investigation rationale is provided in the Discussion Section.

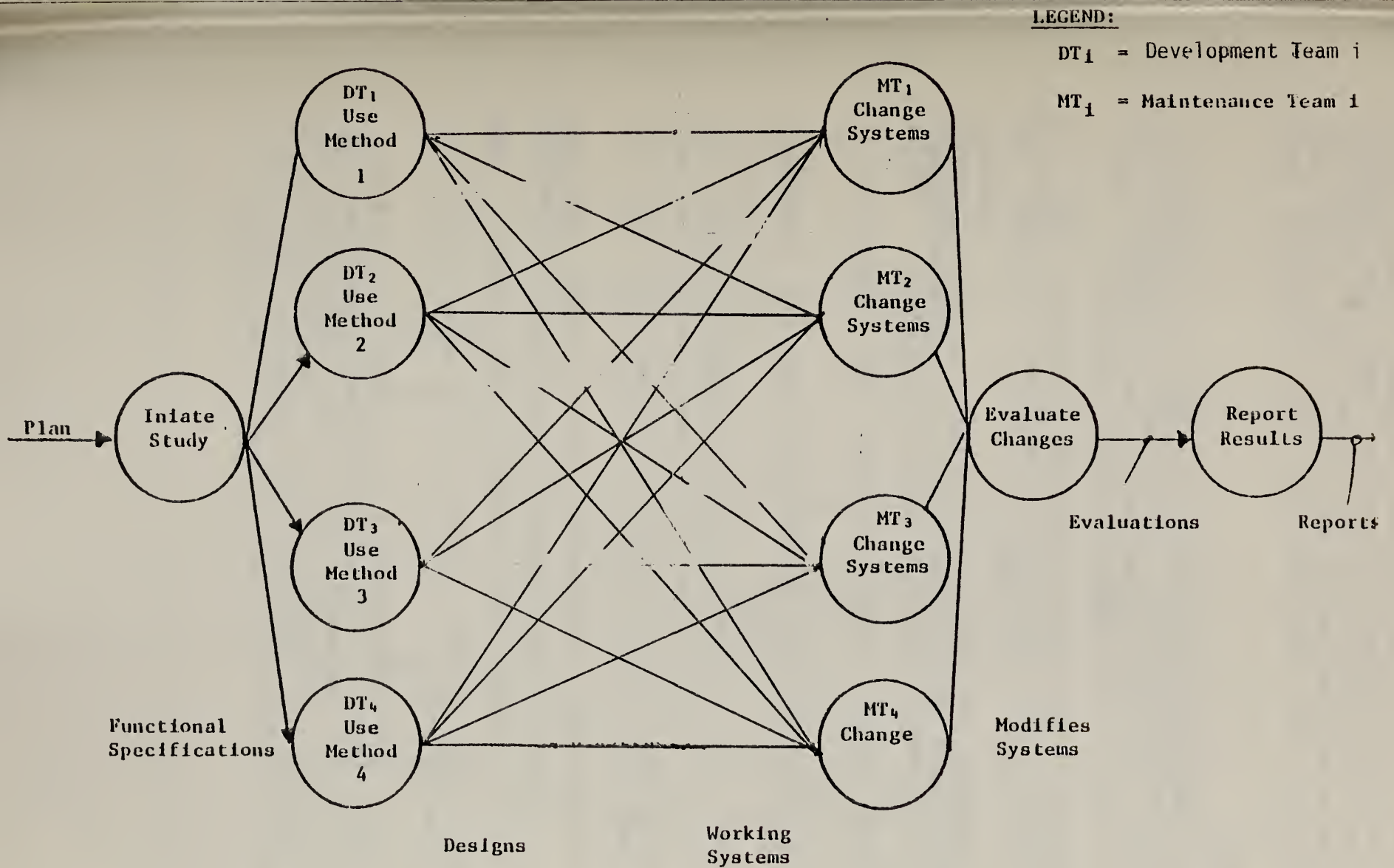


Figure 1: Comparative Study Information Flow

STUDY PLAN

This section presents the proposed comparative study plan in detail. The overall project is broken into five phases of activity. Figure 2 indicates these major phases and shows the major project personnel involved in each phase. The presentation first discusses each major participant and then describes each phase of activity.

Prime Contractor

The entire comparative study must be under the management control of a single organization; the substantive control of the project, however, will be shared with the advisory board. Specific characteristics and responsibilities include:

The contractor must be familiar with DoD procedures and (in general terms) the software development methods and philosophies involved in the study.

The contractor must not have any proprietary interest in any of the development methodologies in the study and cannot supply any of the other personnel (e.g. IV&V team).

The contractor should provide support and administrative direction to the advisory board.

The contractor must make this general plan specific by setting milestones, delivery dates, etc.

The contractor must insure that all requested data is collected.

The contractor must monitor and be responsible for the successful completion of the study.

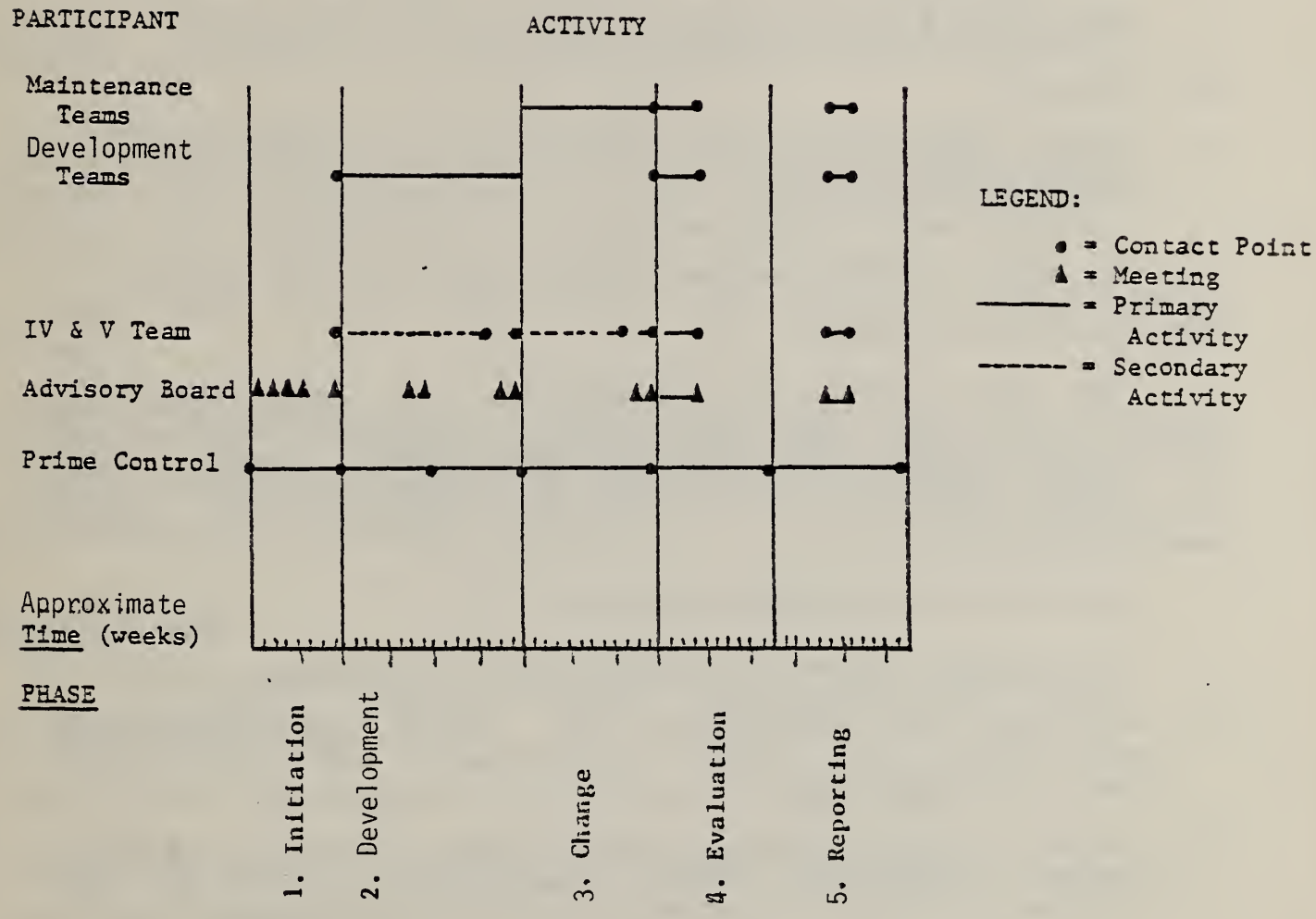
The contractor must draft the final technical summary of the study.

The contractor must publish and disseminate the results.

The contractor must provide liaison with other AJPO and Ada-community activities.

Precise details of subcontracting are not a concern except that it must be noted that: a) the prime contractor must have the authority as well as the responsibility to make sure that the study is successfully completed, and; b) contracting details must not be allowed to interrupt or interfere with the substantive work in any way that would prejudice the results.

Figure 2: Study Plan



Advisory Board

The prime contractor will have responsibility for overall control of the study. A high-level advisory board will share responsibility for substantive technical decisions. We suggest a board of fewer than 10 persons composed of one or more members from:

1. Ada Joint Program Office
2. prime contractor (technical person not under the control of the project manager);
3. defense contractor (not otherwise involved in the study and with no financial interest in any of the development methodologies);
4. non-defense system developer;
5. academia;
6. the Ada community at large (chosen in as representative a fashion as possible).

All members should be technically competent and capable of substantively reviewing and contributing to the work being conducted in the study.

The Board should be financially and administratively supported by the prime contractor. A specific budget should exist for obtaining specialized assistance (e.g. someone expert in running studies of this type) if the Board decides it needs additional information in order to fulfill its review responsibilities.

The duties of the Board include:

1. reviewing and approving the detailed project plan;
2. reviewing and approving the choice of development methodologies and teams, implementation teams, maintenance teams, and IV&V teams;
3. reviewing and approving the problem statement, problem change, maintenance changes, measurements to be taken, IV&V procedures, and all other major technical decisions;
4. making an independent evaluation of the data and presenting a written report to the prime contractor for inclusion in the final report;
5. reviewing and approving the release of the final technical summary of the study;

6. writing and publishing an independent evaluation of the study's methodology (to permit improvement of future studies).

Development Teams

There will be one development team per methodology chosen for study. Wherever possible, the development team shall include the developer of the methodology. The organization (number of people, management structure) of each development team is determined by that team, subject to the requirements and constraints noted below in the description of each phase of work.

A development team shall not employ more than one method or participate in more than one development effort that is a part of this study. No member of a development team shall participate in any other phase of the study except as noted.

Development teams will be responsible for implementing each design in Ada on a common machine. Each development team shall have members that are knowledgeable in Ada. Each team shall be responsible for delivering the maintenance manual (see Appendix 4) in addition to carrying out the coding and checkout.

IV&V Team

This team should be independent and unbiased. Its responsibility is to ensure that the workproducts produced meet the project standards. Although a single IV&V contractor should be used, it is likely that separate teams will be assigned to each development and maintenance team.

Maintenance Teams

The maintenance teams are a critical factor in the study and must be carefully chosen. Their responsibilities and characteristics include:

1. making specified changes to all systems in the study;

2. updating maintenance manuals to reflect the changes;

3. recording required data;

4. recording subjective observations on the ease of change in each system;

Members of maintenance teams should be representative of personnel that might be found in an Ada application situation:

1. they should be competent in Ada coding, but not experts;

2. they should not have a detailed knowledge of any specific development methodology;

3. they should have experience in reading and understanding software documentation;

4. they should have significant experience in maintaining software.

The attempt is to simulate a maintenance group that may be called upon to modify Ada code in an embedded system without having participated in the design or coding process, working only from the maintenance manual.

Project Phases

For each phase, we will define the primary focus and list the major deliverables, activities, and considerations.

Phase 1 - Initiations

Focus: Detailed planning and initiation of study.

Deliverables: Detailed project plan, detailed technical requirements, contracts with participant teams and advisory board.

Activities:

1. form advisory board;

2. select and contract with all participant teams; the number of development methodologies to be selected should be four or fewer;

3. refine statement of problem(s);

4. refine measurements to be taken;

5. outline final report;

6. establish secondary information to be collected;

7. establish IV&V procedures;

8. establish study monitoring mechanisms;

9. establish liaison with other AJPO and Ada-community projects.

Considerations:

The advisory board should be involved from the start and assist in making all technical decisions.

It is advisable to choose dissimilar development methodologies, rather than choosing possibly competitive ones.

The problem stated in Appendix 2 is of moderate size so that the developer can illustrate the development methodology clearly, and so that others can comprehend the basic concepts of the methodology. Ideally, the methodologies should also be applied to a design problem of significant size, e.g., several months of design work, which is more characteristic of large-scale embedded systems applications. However, time and cost considerations may make it infeasible for the methodologies to be applied to a large scale problem, and the results from the medium-sized problem of Appendix 2 may have to suffice.

Every care should be exercised to review the functional specfications thoroughly before development begins to prevent changes or misunderstandings. It is suggested all development teams, the advisory board, and the prime contractor meet before development begins.

Phase 2 - Developments

Focus: Design and implementation of system by different development teams to satisfy Problem Statement (4 different methodologies)

Deliverables: One implementation by each team, maintenance manual, IV&V report, measurements, observations, and other software documentation as required by the methodology.

Activities:

1. technical design, detailed design, and coding by each development team;

2. response to a change in requirements;

3. observation and data collection on development activities;

4. verification that each development product (design, detailed design, code) meets the functional specification;

5. production of maintenance manual;

6. IV&V of development activities.

Considerations:

An upper bound on the time for development should be established, but each development team should have sufficient time, in the team's judgment, to produce as good an implementation as possible using the methodology. Modifying this consideration, however, is the requirement that the software should be produced in a straightforward manner with only normal time permitted for review and rework. Specifically, complete redesign, recoding, or excessive polishing should not to be permitted. The advisory board should determine appropriate limitations.

The change in requirements should be determined by the advisory board and the prime contractor at a meeting to be held a few weeks after the start of development. It should be a typical change that is representative of those that occur in DoD. The actual change should not be known prior to this meeting. The change must be issued simultaneously to all development teams by the prime contractor.

Complete data collection and observational requirements and techniques must be established in advance -- suggestions are made in Appendix 3.

An IV&V team member should be assigned to each development team to make independent observations and to relieve developers of the burden of data collection (insofar as possible). IV&V should not interfere with the progress of the development.

Any questions or changes regarding the problem requirements that arise during the development must be handled explicitly by the project monitor. Questions should be transmitted in writing to the prime contractor, who should prepare a written reply, with the advice of the advisory board if necessary. All replies must be distributed to all development teams.

Each development methodology is assumed to provide its own definition of what information results from applying the method (and in what form). This definition must be made explicit and reviewed by the advisory board to insure conformity with the information outlined in Appendix 4, Section 3.

The study is not a race between methods. Rather it is a comparison of development methodologies of different types.

The IV&V function is intended primarily to make sure that the study requirements have been fulfilled, not to check the technical accuracy or quality of the design.

Any major inconsistencies in the development or correct operation of the code that is found by the IV&V team must be corrected by the development team before any software is

turned over to the maintainers.

The maintenance manual is outlined in Appendix 4. It should contain all information about the software needed by the maintainers. This information should describe the design representation (charts, etc.) so that both the design documentation and the code can be maintained.

Standard Ada coding must be employed in the implementation.

Coding style guidelines should be established and followed.

A standard test should be established so that each implementation can be checked.

The maintenance manual should be completed, and verified to be complete, for the maintenance activity.

Phase 3 - Change

Focus: Change of each system to meet a specific set of change requests.

Deliverables: Changed systems, IV&V maintenance reports, measurements and observations.

Activities:

1. code changing;
2. documentation changing;
3. observation and data collection on maintenance activity;
4. IV&V of changes.

Considerations:

It is critical and essential that the development and maintenance teams not know the nature of the changes before the initiation of this phase. It is suggested that the advisory board develop the set of changes.

As with the implementation, the changes are to be made and tested for accuracy against a standard set of tests.

The IV&V effort should verify the changes and validate the system operation.

All changes should be reflected in the code, the maintenance manual, and all other affected documentation.

Phase 4 -- Evaluation

Focus: Evaluation of the impact of the different methodologies on maintenance from several different perspectives.

Deliverables: Individual evaluation reports, final report.

Activities:

1. evaluation of the correctness of changes to the software by each development team;

2. evaluation of the change process by the maintenance teams;

3. evaluation of the changes by the advisory board;

4. collection and summary of individual reports by prime contractor.

Considerations:

Individuals and teams involved in the study will perform an evaluation and write a report on their perceptions of the relation between the maintainability of each system and the development methodology used to develop it.

The prime contractor has the responsibility for going beyond these individual reports, to generalize from all of the data, observations, and individual evaluations collected in earlier phases. The advisory board will assist in refining this final report.

Phase 5 - Reporting

Focus: Distribution of results to the technical community.

Deliverables: Distribution of reports, successful completion of conference.

Activities:

1. widespread distribution of report;

2. conference to present and discuss results;

Considerations:

The results of this study will not be definitive and will be open to interpretation in most instances. It is essential that an open forum be held to permit discussion of the results, questioning of the methods used, and investigation of the data collecting techniques.

DISCUSSION

In this section we provide rationale for some aspects of the study presented in the previous section and discuss several alternatives.

Overall Structure

The overriding principle in devising the entire study is simplicity. It is felt strongly that careful investigation of a specific question can be more valuable than a more broadly-based study providing less detailed data. This position is supported by the impact of Dijkstra's use of structured programming and levels of abstraction in the development of the T.H.E. operating system [E.W. Dijkstra, "The Structure of the 'THE' Multiprogrmming System," CACM 11,5 (1968), pp. 341-346], Baker and Mills' use of structured programming technology in the New York Times experiment [F.T. Baker, "System Quality Through Structured Programming," Proc AFIPS 1972 FJCC, pp. 339-343], and Parnas' use of information hiding in the design of his KWIC system [D.L. Parnas, "On the Criteria to be Used in Decomposing Systems into Modules," CACM 15,12 (1972) , pp 1053-1058]. None of these cases yielded statistically significant results and the results were open to interpretation. Yet they have had an extremely strong impact on the thinking of software developers and have been widely adopted. A large measure of their impact was due to their narrow focus, so that one associates project success with the particular new items of technology being employed.

It is not anticipated that there will be a strong differentiation between the various development methodologies employed. Indeed, it would not be surprising if the final conclusion is that the use of any explicit development methodology produces significant improvements in maintainability. While many believe this already, a convincing demonstration would go far toward facilitating the adoption of improved methods.

The conduct of this study is intended to provide a clear model of the use of good software engineering project principles. This is part of the motivation for use of IV&V, external reviewers, and other devices.

Prime Contractor

A prime contractor was specified to maintain fidelity to the way business is done in the Ada community and to have a single point of control and responsibility for the project. Most of the details of the study are in outline form to permit the prime contractor to refine the plan to fit the specifics of the actual investigation. This, coupled with the responsibilities of the prime contractor, means that the organization fulfilling that role must be highly capable of providing technical as well as managerial leadership.

Advisory Board

There are three primary reasons for having an advisory board. First, it provides the necessary technical overview for the entire study by performing detailed tracking of the investigation. While the prime contractor could do this alone, it is better to separate, at least partially, the day-to-day contract administrative matters from the technical advice so neither is shortchanged. Second, since this study could have a large impact, it is important to get the very best guidance possible from as wide a segment of the technical community as possible. Third, the board can serve as a resource for the prime contractor.

Development Teams

It would be interesting to have multiple designs based on each method in order to obtain quasi-statistical results. This, however, introduces many more problems (capabilities of different groups, for example) than it solves. As we noted above, we are striving to obtain some results that can be as strongly interpreted as possible. Thus, we have chosen to not address issues such as whether a particular method can be applied in production environments by someone other than its inventor. (All of the methods suggested have previously been successfully transferred.)

IV&V Team

The IV&V team is to mirror the actual development environment that is promoted within DoD work. Furthermore, it provides a simple way to assure that the technical work of the study conforms to plan. Finally, it will introduce an added guarantee of objectivity and, through their observational activities, partially free the development teams from the distraction of record keeping required by the nature of the study.

Maintenance Teams

It is often argued that the best programmers should be assigned to the maintenance activity. However, it has been observed that this rarely happens. Instead, the model has been that maintainers are faced with a broken (or inadequate) piece of equipment. They are given an instruction manual to go with their limited knowledge of the design process that created the piece of equipment. Their task is to fix the equipment. This analogy is used with appropriate modifications to make it fit the software situation.

Project Phases

Phase 1 - Initiation

As noted above this plan will need to be refined at the start of the study to make it fit the actual situation at that time and place. It is urged, however, that changes in structure or intent be made sparingly and only after careful review.

A standard procedure in an investigation of this type is to run a pilot study (perhaps on only one method) to help refine the details of this plan. It is recommended that this be done in this case. Methodological experimentation in general and especially in the software area is still at a rudimentary stage. Thus, it is impossible to specify accurately data collection procedures, establish control and coordination mechanisms, and foresee other problems. With a pilot study, the refined investigation can be reasonably assured of success; without one, there is a danger of a flawed investigation.

The problem selection -- an electronic mail system comparable to the UC Berkeley mail system developed in the Computer Systems Research Group -- was influenced by several considerations, including the following:

1. a realistic problem of moderate size, i.e., nontrivial, but not huge;

2. limited expert knowledge needed;

3. a problem involving real time processing and potential concurrency;

4. suitable for programming in Ada, involving Ada features such as tasks, packages, and exception handling;

5. balance between process design and data design;

6. a number of different possible solutions.

Outlining the final report is required during project initiation. It is expected that this activity will force those people responsible for its content (the prime contractor and the advisory board) to precisely specify what information they must collect during the investigation.

As noted above, there is only limited experience with measurement of parameters of interest in this type of investigation. This experience, though, shows that the measurement activities should be planned from the outset and that ample time and resources should be allocated for data collection during the investigation.

Phase 2 - Development

It is important to try to strike a balance between letting developers have as much time as needed to produce "perfect" software (necessary if we want an accurate test only of the technical capabilities of the method) and reality in which there is never enough time for perfection.

Making observations and collecting data in this type of investigation takes a surprising amount of time and effort. Since this normally would not be a part of a development effort (it should be, but it usually isn't) these activities should not interfere with the development any more than necessary. The solution is the introduction of an IV&V team member as an adjunct to each development team (this should be done during maintenance as well) to help with the data collection. All of the teams must expect to devote some effort to data collection, however.

Change during the development effort is inevitable since formal specifications that can be interpreted in different ways are being used. Careful review of the problem statement and a face-to-face meeting between all developers and the project management at the beginning should serve to prevent many of the problems. For those that do arise, it is essential that everyone have the same information so that implementation won't differ because of differences in knowledge about the problem statement.

Maintainers of systems rarely have the luxury of direct contact with the developers -- that is one of the factors that makes maintenance so expensive. The critical issue in this investigation is to determine the impact of various methods on the maintenance of the resulting system. The impact may come through two channels -- via improved structure in the design itself and via improved structure in the documentation.

For most (if not all) of the methods that may be studied, competent programmers should be able to understand the results of applying a methodology with only a minimum of training (to be supplied by the documentation itself). This same minimal training should permit the maintenance team not only to use the documentation to help speed up and improve the quality of the maintenance activity but also to record their changes in altered editions of the documentation. An interesting point raised by one researcher (Rob Kling) is that the work style of the design teams may have more influence than the method they use. While this may be true, we do not know how to control for it and can only warn the investigators to be on the lookout for such effects.

An alternative to the entire structure of this investigation would be to have construction teams implement a design developed by design teams (as was defined in the original version of this study). This would provide information relating to the effectiveness of alternative design methods, however, it adds considerable complexity and expense.

Phase 3 - Change

More than one maintenance team, representing different organizations, should be chosen. Each maintenance team should have exactly the same task -- to prepare a set of revisions to each of the implementations.

Phase 4 - Evaluation

Evaluation should be performed by as many different people as possible since the subjective evaluations will be an important adjunct to the data collected. Indeed, if several more or less independent groups concur in their subjective evaluations, then this will strengthen (or weaken) the objective data that is collected. It will also help make the results more believable to the community.

Phase 5 - Reporting

The purpose of holding a workshop to present the results is not to argue about them, but rather to permit the community to probe how the results were obtained in as much detail as desired. This will be beneficial in exposing weaknesses that can be corrected in future investigations as well as making the results more believable (because they are open to inspection) to the community.

FURTHER STUDIES

Because software development is a relatively new field, it would be superfluous to outline several possibilities here (a recent issue of Software Engineering Notes [vol. 7, no. 1] contains several concrete proposals), but the value of replicating this (or any) investigation several times in different circumstances would be notable.

However, two things need to be emphasized. First, objective investigations of the ways in which software is developed must be carried out to permit us to make more rational decisions regarding the way we organize and carry out system development. Second, the investigative process must be refined greatly from its present state.

ACKNOWLEDGMENTS

The work was aided by the results of the ACM SIGSOFT Symposium on Tool and Methodology Evaluation, as published in Software Engineering Notes, vol. 7, no. 1 (January, 1982), pp. 6-74. Comments given on the initial draft of this document by Ruven Brooks, Bill Curtis, Rob Kling, and Ben Shneiderman were appreciated. Those of Deborah Boehm-Davis and Elizabeth Kruesi were especially detailed and instrumental in preparation of the final draft. Susan Richter did the artwork and assisted with the

text editing and formatting. Comments affecting the edited version of this report were provided by Elizabeth Kruesi, Barry Boehm, J. A. McDermid, John Mellby (and other unnamed software managers at Texas Instruments), and D. Lefkovitz.

APPENDIX 1 - Suggested Methods for Study

We have identified upwards of 40 methodologies for software development and have sent questionnaires to the creators of these methodologies. (See "Ada Methodology Questionnaire Summary," by M. Porcella, P. Freeman, and A.I. Wasserman.) While we may have overlooked some approaches, we believe that we have identified those that have been most widely used.

In selecting methodologies to be compared, we have used four subjective criteria:

1) widespread use

2) publicly available documentation

3) use of the method within DoD

4) mapping of design or specification primitives to Ada

However, not all of the recommended methodologies satisfy all of the criteria.

We also have sought to identify methods spanning a range of formalism. On that basis, we recommend that the comparative study choose from the following methodologies:

1) STRADIS (McDonnell Douglas)

2) Yourdon

3) Jackson Design Method

4) HOS (Higher Order Software)

5) ISAC (Mats Lundeberg)

6) SADT (SofTech)

7) SARA (G. Estrin, UCLA)

Other potential candidates if broader diversity is desired, are:

1) HDM (SRI International)

2) Wellmade (Honeywell)

3) User Software Engineering (A.I. Wasserman, U.C. San Francisco)

4) Warnier/Orr (Ken Orr & Associates)

5) PSL/PSA (D. Teichroew, U. of Michigan)

APPENDIX 2 -- Problem

This problem is divided into two parts: an initial problem, which should be designed and coded in Ada, and a set of requested changes that should be given to a maintenance team for redesign and code modifications.

This is a moderate sized problem, exhibiting elements of real time processing and concurrency, and necessitating both process and data design. It is suitable for programming in Ada, and represents a problem of realistic interest to the Ada community. Although it is not in itself an embedded system, it could easily be part of an embedded communication system.

The moderate size is intended to strike a balance between a trivial problem that can be solved without the aid of any development methodologies and the mammoth design problems that characterize many of the larger command-and-control systems.

If time permits, the prime contractor could construct a much larger design problem for the development teams. However, we do not believe that such an exercise would be cost effective in showing the efficacy of development methodologies.

The Problem Statement

We wish to design and build an electronic mail system for a distributed setting. The setting consists of one or more local networks, each of which are known by an alphanumeric name of 1 to 8 characters, e.g. berkeley. These local sites are connected by common carrier to the ARPA network.

Each local site has one or more machines. If a local network has more than one machine, each is named by a single letter (upper or lower case).

Every machine has one or more users, each of whom has a login code of 1 to 8 alphanumeric letters, e.g., druffel. Each login name is unique on a machine, but the same login name may exist on any number of machines within the entire system. Thus, an individual may have any number of login names, not necessarily the same, on any number of machines at any number of sites.

Valid addresses may therefore be described by the following syntax:

```
[machine ":"] loginname ["@" site]
```

The following constructs are thus permissible:

```
freeman
jones@dec
v:wasserman
```

A:good@texas

There are, within the network, a known set of sites at any time. Furthermore, there is a known set of local machine identifications at each site.

The command

mail addressee

will then accept text to be sent to the addressee at the designated site and machine. Input is accepted until the input stream receives an EOF character (ctrl-D). An alternative form is to transmit a file to the addressee. This is accomplished with the command

mail addressee < fname

where fname is a file for which the user has read/copy permission.

If mail cannot be sent to addressee, the message is returned to the person sending the message, preceded by the text

Could not send mail to "addressee" -- text follows

It is then seen by the user as a message sent to oneself.

When a user logs on a given machine, the system will print the message

You have mail.

if mail has arrived for the user (according to that login name for that machine) since that user's previous login. (Note: if necessary for your solution, you may assume that transmission of the message occurs within a day, but is not necessarily immediate.)

The user can then access the mail by typing the command

mail

with no parameters. If there are no waiting messages (the user simply typed the mail command in the absence of the "You have mail" message), the system will reply

No mail.

Otherwise, the system will produce a list of unread messages, in the following form

N sender date lines/chars

where N is an integer, sender is the login name and site of the

sender of the message, date is the time that the message was received, lines tells how many lines are in the message, and chars gives the number of characters in the message. (Note that untransmitted messages, as described above, are returned in this manner.)

Thus, the list might appear

```
    1  freeman@eclb         Sep 12  13:50   25/1204
    2  wasserman@berkeley   Sep 12  14:25   40/1723
```

The user may then issue the following commands:

N	typing a number causes the message to be printed; N must be a valid message number or a diagnostic will be printed
dN	causes message N to be deleted
s fname	causes most recently printed message to be stored in a local file named fname
h	prints the header line for all unprocessed messages
q	quit, saving any unread messages for the next login

Thus, the following sequence of commands would cause message 1 to be printed and deleted, and message 2 to be printed and saved in file berk.12Sep before quitting.

```
1
d1
2
s berk.12Sep
q
```

Sample Modifications

We wish to make the following set of modifications to the system described above. The result of this set of changes should be a revised design document and revised code, along with supporting documentation showing the affected modules and the relevant design decisions. Wherever possible, the module changes should be linked to the changed requirements.

(1) Allow user names to 1 to 10 characters instead of 1 to 8 characters.

(2) Permit a message to be sent to more than one user at a time. This may be accomplished in two different ways:

a) mail addresseelist

where addresseelist is a sequence of (1 or more) addressees with names separated by commas, e.g., mail a:smith, J:williams, casey@bat

b) mail alias

where alias is a predefined string in a file of aliases that creates an equivalence between the alias and a list of addressees.

Thus, the entry

```
friends = a:smith, J:williams, casey@bat
```

would cause the command

```
mail friends
```

to have the same effect as the command in part a).

(3) Add the reply command (r) to permit the user to reply directly to the most recently processed message. Thus, if one had read a message from freeman@eclb, the command

```
r
```

would accept input from the user and transmit it to freeman@eclb. Similarly, the command

```
r < fname
```

would transmit the contents of file fname to freeman@eclb.

(4) Add the list command (l) to permit the user to obtain a list of all unprocessed messages in the same format as is given in response to the original mail command.

APPENDIX 3: Observational and Data Collection Requirements

NOTE: These requirements are only in outline form at this point. They must be expanded and refined before the actual study is carried out. A good reference for additional suggestions on data collection is Part V of *Tutorial: Models and Metrics for Software Management and Engineering* by Victor R. Basili (IEEE Computer Society).

We suggest the following information, at least, be collected at each stage:

DESIGN

- team information
  - names of all participants
  - technical background of each
- environment description
  - working conditions
  - on-line aids
- technical activity
  - accurate record of actual hours worked by individual
  - breakdown into categories, including
    - background research
    - meetings
    - working group sessions
    - individual technical work
    - documentation (low creative component)
    - non-project activity (interruptions, etc.)
- *** where possible, measurements should be on 1/4 hour breakdown
- technical results
  - number of modules
  - number of distinct data structures
  - pages of documentation
  - characters of documentation

CONSTRUCTION

- team information as above
- technical activity as above, but modified to include
  - coding
  - waiting for system
  - correcting compile errors
  - interpreting design
  - seeking clarification of design
- results to include
  - number of Ada statements
  - number of declarations
  - number of characters in Ada representation

CHANGE

team information

technical activity
- time spent understanding change requests
- time spent understanding design
- time spent understanding code
- time spent making changes to code
- time spent making changes to design
- time spent compiling changes

technical results
- lines of code changed
- number of modules changed

APPENDIX 4: Maintenance Manual Outline

The Maintenance Manual must contain all information that the maintainers of a system need in order to make changes efficiently. The precise content of the information will depend on the development methodology used, but the format of the document should be approximately the same for each method.

1. SYSTEM OVERVIEW

 1.1 Basic purpose of system
 1.2 Functional description of operation
 1.3 Implementation considerations
 1.4 Operational considerations
 1.5 Guide to Maintenance Manual

2. SYSTEM SPECIFICATION

 2.1 Complete definition of all external functions
 2.2 Complete definition of all external data items
 2.3 Complete definition of all internal functions
 2.4 Complete definition of all internal data items
 2.5 All constraints (design, implementation, operation)

3. SYSTEM STRUCTURE

 3.1 Overview of system modularization
 3.2 Logical data definitions
 3.3 Module Interface definitions
 3.4 External module definitions

4. IMPLEMENTATION

 4.1 Detailed design of modules
 4.2 Physical data structures
 4.3 Code listings

APPENDIX 5 -- SADT Model of Project Plan

A complete SADT model consists of two kinds of diagrams: activity diagrams (called actigrams) and data diagrams (called diagrams). The view of an actigram is that data objects flow between activities while the view of a datagram is that activities during their operation access data objects. The only difference is the center of attention. Only actigram models will be discussed in this appendix.

THE ELEMENTS OF AN ACTIGRAM

An actigram depicts three to six activities which are represented as boxes. The limit on the number of activities depicted helps to limit the amount of information a reader of an actigram must deal with. The boxes of an actigram are connected by arrows which represent data objects. Actigrams are data-flow diagrams. This means that the activity of a box takes place only when the data objects represented by incoming arrows to a box are present.

The positions of the arrows on the box determines what type of data an arrow represents as shown in Figure 5.1. When input, control, and mechanism objects are present, the activity uses the mechanism as an agent to transform the input data objects into the output data objects under the guidance and constraints of the control data objects. Activity names should be verbs, while data object names should be nouns. Each activity must have at least one control and output.

A double headed dotted arrrow may be used as a shorthand in SADT to denote data relations between activities as shown in Figure 5.2.

THE STRUCTURE OF AN SADT MODEL

Each actigram is an elaboration of an activity box in a higher-level diagrsm called the parent diagrsm. If a page number appears in parentheses just outside the lower righthand corner of an activity box, then this number specifies the page of the actigram which elaborates the box. The inputs, outputs, controls, and mechanisms used in an actigram are the same as those as those on the corresponding activity box in the parent diagram. Each actigram should include from three to six activity boxes.

The highest-level actigram of a model is the only exception to the three to six activity rule and it presents only one activity, the one being modeled. The inputs, outputs, controls, and mechanisms which are used in the rest of the model are specified on this highest-level actigram called A-0. The A-0 actigram represents the context in which the system being modeled operates. As a part of the context the A-0 actigram explicitly

states in prose the purpose of the model and from what viewpoint the model was made.

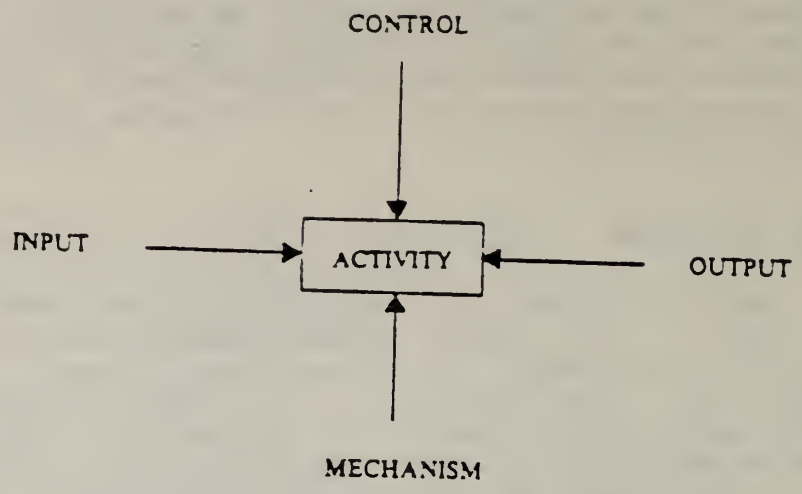


Figure 5.1.

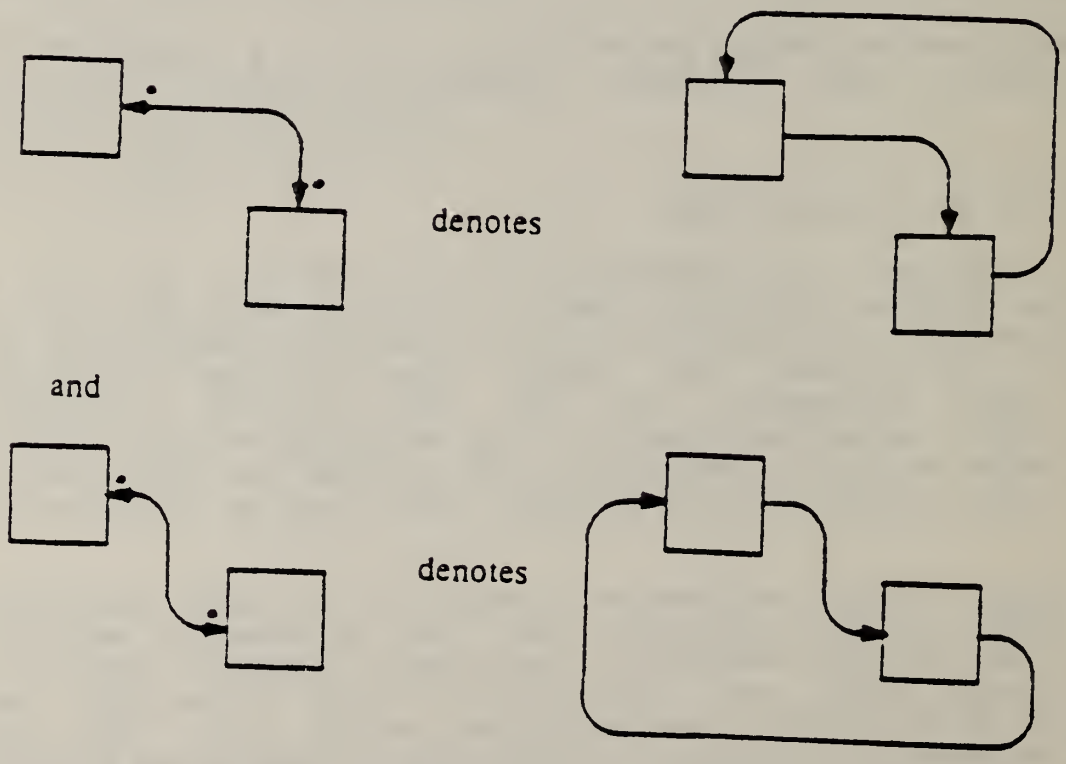
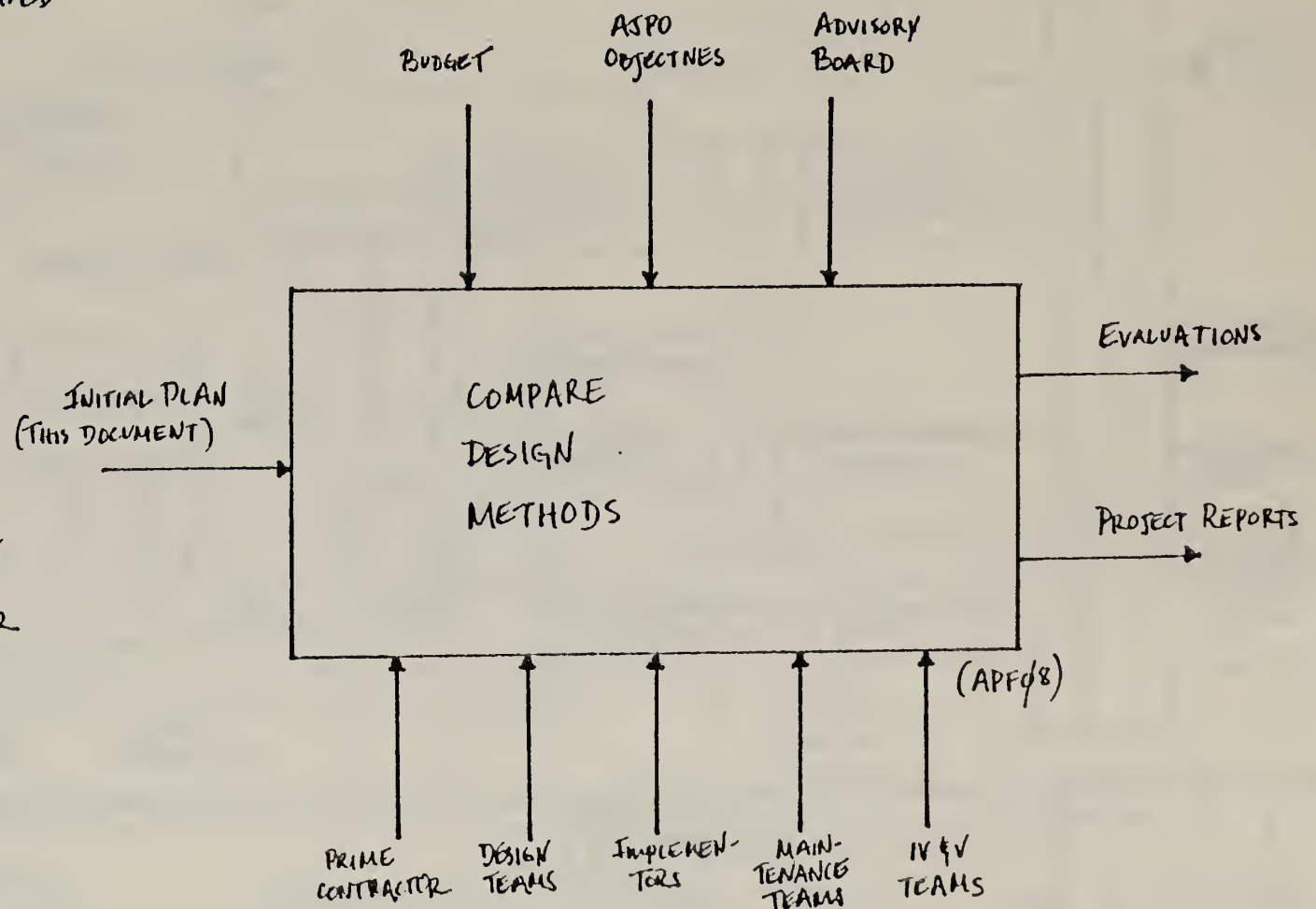


Figure 5.2.

USED AT:	AUTHOR: PETER FREEMAN PROJECT: ADA METHODS	DATE: 10/24/82 REV:	WORKING DRAFT RECOMMENDED PUBLICATION	READER DATE	CONTEXT:
	NOTES: 1 2 3 4 5 6 7 8 9 10				

[1] GLOSSARY NOT YET PREPARED FOR THIS MODEL



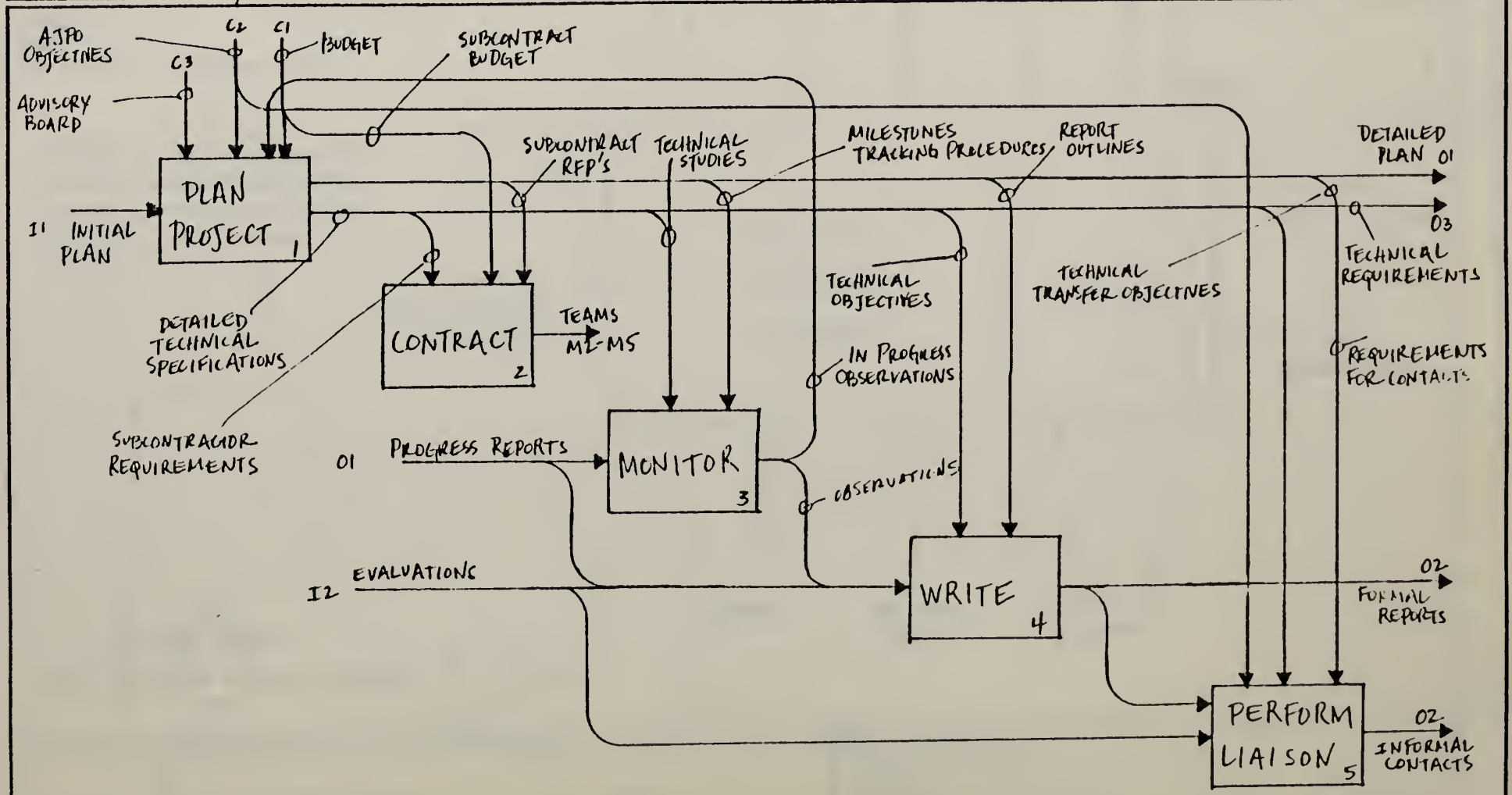
PURPOSE: PROVIDE MODEL OF DESIRED PROJECT ORGANIZATION TO PERMIT ACCURATE, DETAILED PLANNING BY PRIME CONTRACTOR

VIEWPOINT: PREPARERS OF INITIAL PLAN

NODE: A-0	TITLE: COMPARE DESIGN METHODS (CONTEXT)	NUMBER: APF07 (APF/S)

SADT® DIAGRAM FORM ST098 9/75

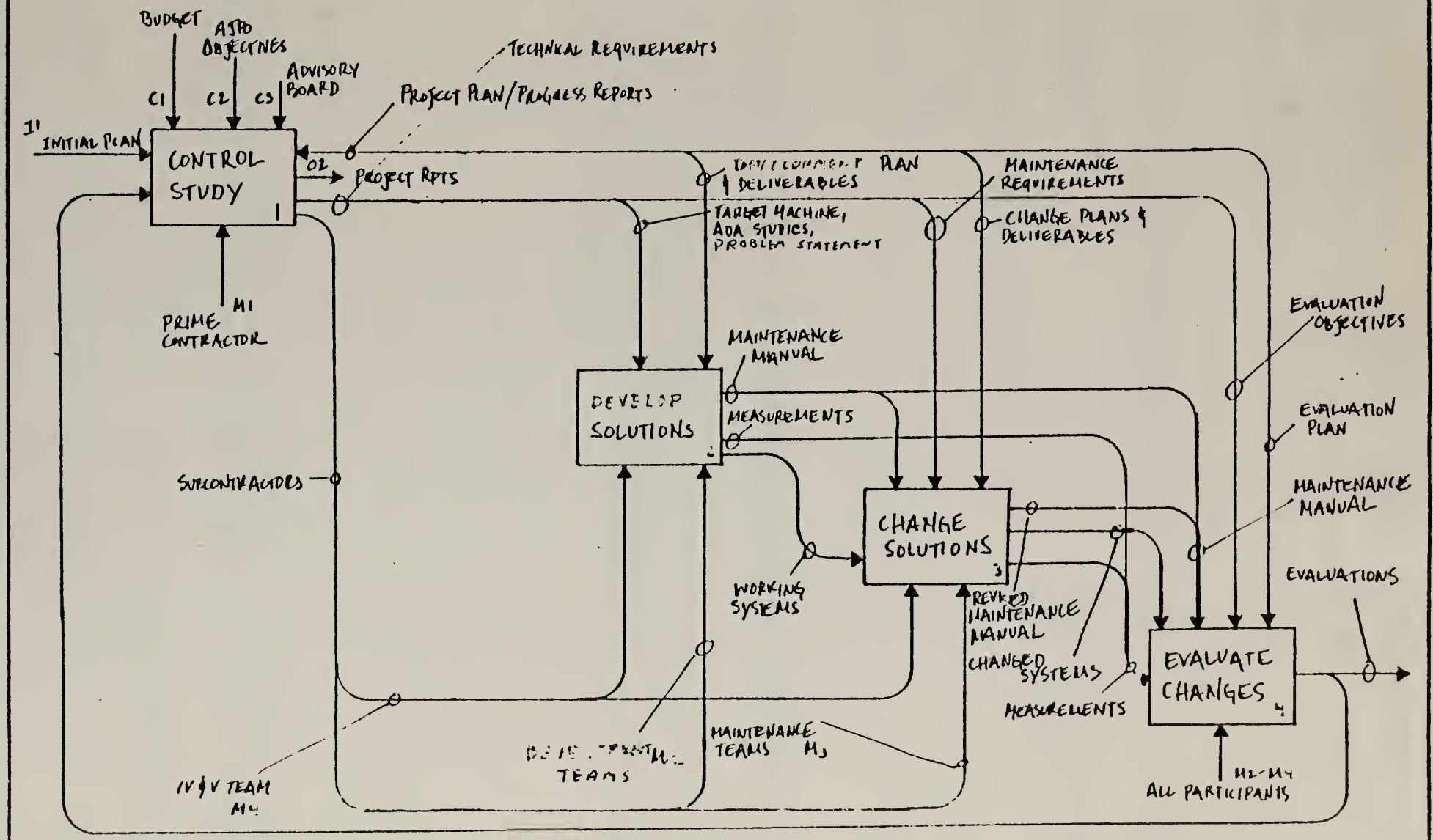
USED AT:	AUTHOR: PETER FREEMAN PROJECT: ADA METHODS NOTES: 1 2 3 4 5 6 7 8 9 10	DATE: 10/24/82 REV:	WORKING / DRAFT / RECOMMENDED / PUBLICATION	READER / DATE	CONTEXT:



NODE: A1	TITLE: CONTROL STUDY	NUMBER: APE09 (APE06)

SADT™ DIAGRAM FORM ST098 9/75

USED AT:	AUTHOR: PETER FREEMAN PROJECT: ADA METHODS NOTES: 1 2 3 4 5 6 7 8 9 10	DATE: 10/24/82 REV:	WORKING DRAFT RECOMMENDED ☒ PUBLICATION	READER DATE	CONTEXT:



NODE: A0	TITLE: COMPARE DESIGN METHODS	NUMBER: APF08 (APF04)

NBS-114A (REV. 2-8C)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET (See instructions)	1. PUBLICATION OR REPORT NO. NBSIR 84-2827	2. Performing Organ. Report No.	3. Publication Date March 1984

4. TITLE AND SUBTITLE

Comparing Software Development Methodologies for Ada: A Study Plan

5. AUTHOR(S)

Peter Freeman and Anthony I. Wasserman(Univ. of CA), Raymond C. Houghton, Jr., Editor

6. PERFORMING ORGANIZATION (If joint or other than NBS, see instructions)

NATIONAL BUREAU OF STANDARDS
DEPARTMENT OF COMMERCE
WASHINGTON, D.C. 20234

7. Contract/Grant No.

AJPO-83-27

8. Type of Report & Period Covered

Final Report

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (Street, City, State, ZIP)

Department of Defense
Ada Joint Program Office
3D139 (400AN), The Pentagon
Washington, D.C. 20301

10. SUPPLEMENTARY NOTES

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)

A study plan is presented that concentrates on the impact of alternative development methodologies on the maintainability of Ada code. The basic elements of the study include: (1) experts in each of several methods create Ada implementation for a specific problem, (2) each implementation is modified by each of several maintenance teams, and (3) the impact of the methodology on the maintainability of the resulting Ada-coded systems is evaluated and reported.

12. KEY WORDS (Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)

Ada; software design, software development; software engineering; software maintenance; software methodologies.

13. AVAILABILITY

☒ Unlimited

☐ For Official Distribution. Do Not Release to NTIS

☐ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.

☒ Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES

39

15. Price

$8.50