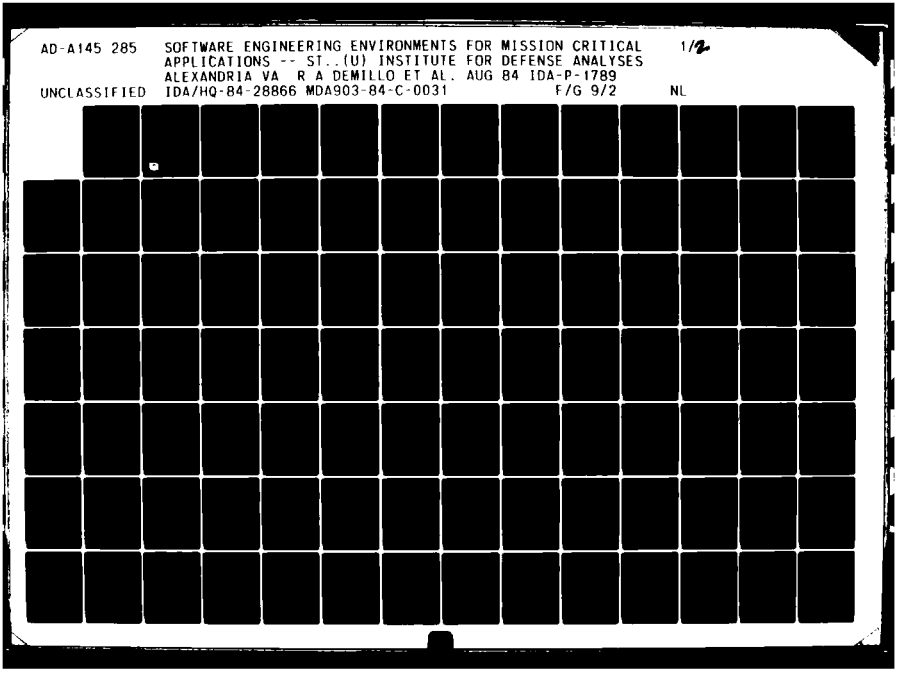
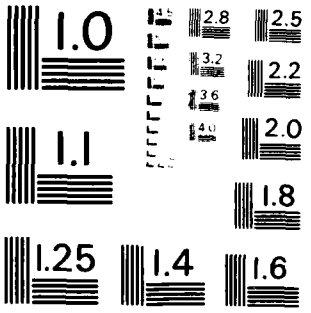
AD-A145 285 SOFTWARE ENGINEERING ENVIRONMENTS FOR MISSION CRITICAL 1/2
APPLICATIONS -- ST..(U) INSTITUTE FOR DEFENSE ANALYSES
ALEXANDRIA VA R A DEMILLO ET AL. AUG 84 IDA-P-1789
UNCLASSIFIED IDA/HQ-84-28866 MDA903-84-C-0031 F/G 9/2 NL





12

IDA PAPER P-1789

# SOFTWARE ENGINEERING ENVIRONMENTS FOR MISSION CRITICAL APPLICATIONS -- STARS ALTERNATIVE PROGRAMMATIC APPROACHES



Richard A. DeMillo
Ann B. Marmor-Squires
Samuel T. Redwine, Jr.
William E. Riddle

August 1984

*Prepared for*
Office of the Under Secretary of Defense for Research and Engineering



IDA
INSTITUTE FOR DEFENSE ANALYSES



IDA Log No. HQ 84-28866

The work reported in this document was conducted under contract MDA 903 84 C 0031 for the Department of Defense. The publication of this IDA Paper does not indicate endorsement by the Department of Defense, nor should the contents be construed as reflecting the official position of that agency.

This paper has been reviewed by IDA to assure that it meets high standards of thoroughness, objectivity, and sound analytical methodology and that the conclusions stem from the methodology. IDA does not, however, necessarily endorse the conclusions or recommendations that it may contain.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

| REPORT DOCUMENTATION PAGE | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|---|---|
| 1. REPORT NUMBER | 2. GOVT ACCESSION NO. AD-A145285 | 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE (and Subtitle) Software Engineering Environments for Mission Critical Applications -- STARS Alternative Programmatic Approaches | 5. TYPE OF REPORT & PERIOD COVERED FINAL - Dec. 1983- Aug. 1984 |
| | 6. PERFORMING ORG. REPORT NUMBER IDA Paper P-1789 |
| 7. AUTHOR(s) Richard A. DeMillo, Ann B. Marmor-Squires, William E. Riddle, Samuel T. Redwine, Jr. | 8. CONTRACT OR GRANT NUMBER(s) MDA 903 84 C 0031 |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS Institute for Defense Analyses 1801 N. Beauregard Street Alexandria, VA 22311 | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Task T-4-236 |
| 11. CONTROLLING OFFICE NAME AND ADDRESS STARS Joint Program Office 400 Army-Navy Drive, 9th Floor Arlington, VA 22202 | 12. REPORT DATE August 1984 |
| | 13. NUMBER OF PAGES 149 |
| 14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) DoD-IDA Management Office 1801 N. Beauregard Street Alexandria, VA 22311 | 15. SECURITY CLASS. (of this report) UNCLASSIFIED |
| | 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE NA |

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

software engineering environments; military requirements; computer programming; mission critical computer resources; laboratories; industrial procurement; consortiums; software factories; alternatives

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

Central to the STARS (Software Technology for Adaptable and Reliable Systems) Program is automated software engineering environment(s) (SEE) to provide software development and in-service support for DoD systems with mission critical computer resources (MCCR). This paper briefly describes seven previously proposed alternatives for producing SEEs in the near-, mid-, and long-terms and highlights the advantages and disadvantages of each alternative. It reviews (continued)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. Abstract (Continued)

some key issues for selecting alternatives or combinations of alternatives including DoD's needs, the available technical opportunities and choices, the means used to obtain environment(s), and the costs. This paper also makes recommendations for STARS decision making, suggesting an exploration and decision process as well as a number of possible activities. In depth descriptions of the seven previously proposed alternatives appear in an Appendix.

IDA PAPER P-1789

# SOFTWARE ENGINEERING ENVIRONMENTS FOR MISSION CRITICAL APPLICATIONS -- STARS ALTERNATIVE PROGRAMMATIC APPROACHES

Richard A. DeMillo
Ann B. Marmor-Squires
Samuel T. Redwine, Jr.
William E. Riddle

August 1984


INSTITUTE FOR DEFENSE ANALYSES
1801 N. Beauregard Street
Alexandria, Virginia 22311

Contract MDA 903 84 C 0031
Task T-4-236

## FOREWORD

This report addresses programmatic alternatives for a portion of the DoD STARS (Software Technology for Adaptable Reliable Systems) Program. As a key ingredient in advancing the state of software engineering practices for DoD applications, the STARS program must foster the software engineering environments that appear to be the most promising in the near-, mid-, and long-term.

Senior management of the STARS program, Service Computer Resource Managers, and senior scientists and engineers in the DoD have been presented with a range of possibilities and options. Some of these options are the natural outgrowth of ongoing DoD R&D programs. Others have been proposed by industrial sources and trade groups. A key question is which approaches should be pursued, and what are their relative strengths and weaknesses?

The authors first distilled the most common, previously proposed options into seven synopses and these formed the core of a draft version of this document. In describing these options, the following approach was taken.

- brief description of the option
- objections to the option
- responses to the objections
- discussion of option characteristics
- summary.

However, organizing the discussion in terms of these previously proposed alternatives did not form a good basis for analysis and recommendations (a judgement that was also agreed to by reviewers). These synopses in updated form are preserved in the Appendix to this report and are described briefly in the Background Chapter.

The now somewhat shorter main body of the report concentrates on the requirements, issues, and possibilities in the near-, mid-, and long-terms. This allows more natural consideration of the needed mixing and matching of portions of the original seven proposed alternatives. This discussion and the recommendations made are aimed at helping STARS decision makers concerning software engineering environments. However, to more fully review all the arguments, the Appendix should be read as well.

## ACKNOWLEDGMENTS

A number of people provided valuable assistance in the production of this report. The authors are indebted to Paul Clements, Vance Mall, Dan Alpert, Bill Carson, P.M. Davies, Neil Eastman, Jeff Jones, John Manley, Gil Myers, Tricia Oberndorf, Donn Philpot, Brian Schaar, and Bob Wasilausky for reviewing and commenting on the draft. The authors also wish to thank Debbie Franke of the RTI (Research Triangle Institute) for collecting data on the status of industrial environments for the section on Sponsoring Commercial Development and the many people (too numerous to name here) who provided information for this report over the telephone. Sarah Nash was responsible for editing and incorporating changes suggested by reviewers of the draft into the final version. Particular thanks go to Jo Ann Stilley, Carol Powell, Mary Lou Caldwell, Joyce Tuggles, Zelma Cameron, and Charlene Smith for typing the manuscript.

Table of Contents

## EXECUTIVE SUMMARY

### Background

Central to the STARS (Software Technology for Adaptable and Reliable Systems) Program is automated software engineering environment(s) (SEE) to provide software development and in-service support for DoD systems with mission critical computer resources (MCCR). Mission critical software is currently pervasive, complex and critical, and projections are that it will be increasingly so in the future. SEEs are essential to meeting these needs for reliable and maintainable software in an economical and timely fashion.

SEEs must be capable of producing software suited to MCCR applications and adaptable to DoD and Service system development, acquisition, and support policies and procedures. The following MCCR requirements limit the types of environments that are acceptable to MCCR development communities: real-time, high cost of failure, small to very large in size, long life span, extreme support requirements, frequent modifications, high availability requirements, multiple copies of systems, and Government management control. In addition, a viable alternative must promote the rapid, widespread use of the SEE and yield substantial improvements in productivity and reliability. Seven previously proposed alternatives have been identified for producing a SEE in the near-, mid-, and long-terms. There are advantages and disadvantages to each alternative. Important factors in describing and evaluating SEEs are time to deliver, technical risk, design methods used, funding, implementors, ownership, business model, MCCR acquisition strategy, maintainer, and management approach. Figure 1 summarizes these factors for the seven previously proposed alternatives.

The first of the seven, the Joint Service Software Engineering Environment (JSSEE) alternative, has been launched by STARS and is advocated by Service management and a number of Government Laboratories and is compatible with the establishment of a Software Engineering Institute (SEI). It would be built on top of the CAIS (Common APSE Interface Set) virtual operating system and would follow a life cycle oriented methodology. The major objections to this option are that the top-down design approach may limit the environment's flexibility and extensibility, crucial factors in meeting changing requirements of MCCR systems. Careful planning, particularly prototyping, is expected to overcome these problems.

Service Laboratories, working cooperatively or independently, are another alternative for developing a SEE. Although at least one SEE has already been developed by a Navy Laboratory, Laboratories in general have limited resources and were

| | CHARACTERISTIC | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| OPTION | TIME TO DELIVER | TECHNICAL RISK | DESIGN METHOD | FUNDING | IMPLEMENTOR | OWNERSHIP | BUSINESS MODEL | MCCR ACQUISITION STRATEGY | MAINTAINER | MANAGEMENT APPROACH |
| JSSEE | Mid-to long-term | High | Top-Down | Government | SEI or Contractors | Government | Government | GFE | SEI and Services | Centralized |
| Service Laboratories | Near-term to mid-term | Low | Top-Down | Government | Government or Contractors | Government | Government | GFE | DoD Labs or Contractors | Distributed |
| Off-the-Shelf Alternative | Near-term to Mid-term | Low | Bottom-Up | Government to Private | Contractors to Commercial | Government or Proprietary | Government or Commercial | Specify Interfaces | Developer | De-Centralized |
| Sponsoring Commercial Development | Near-term | Low | Bottom-Up | Government | Commercial | Proprietary | Government | Specify Interfaces | Developer | De-Centralized |
| Application-Oriented Environment | Near-term to mid-term | Low | Bottom-Up | Government to Private | Contractors | Proprietary | Commercial | Specify Interfaces | Developer | Distributed |
| Revising Policy | Near-term | N/A | N/A | N/A | Commercial | Proprietary | Commercial | Specify Interfaces | Developer | Distributed |
| Industrial Consortium | Mid-to long-term | Low-High | Top-Down | Private or mixed | Commercial or mixed | Proprietary | Commercial | Unknown | Developer or SEI | Centralized |

Figure 1.

originally established to deal with physical and system-level engineering problems, not software. Creative Laboratory management may be able to solve these problems, making this a low risk (but possibly low payoff) near- to mid-term option.

Modifying existing (off-the-shelf) commercial or Government-sponsored SEEs for MCCR use is an attractive alternative because the technical risk is lower than a "from scratch" development. However, there may be adoption and adaptation problems. There is also concern that selection of this alternative could result in a proliferation of environments.

Identifying and targeting a restricted number of industrial SEEs for Government investment and support is the basis for the Sponsoring Commercial Development alternative. The first step would be to assess the preparedness of industry to participate in such an alternative. A limited survey of seven organizations revealed little development of integrated, advanced SEEs. A more extensive investigation might yield different results. Other arguments against this option are that it would favor some contractors over others and that Government should not be subsidizing commercial developments.

The Application-Oriented Environment alternative is compatible with the other commercial options and involves developing environments with features and capabilities tailored to specific applications. There is concern that environments developed under this option will lack commonality and compatibility, making use of tools in other than their original environment difficult or impossible. Duplication of effort and environment proliferation are other concerns. Development of a plan that recognizes and tries to minimize these potential problems is one response to these objections. Environments would be developed in phases, and public reviews at the end of various phases would select the best candidates to continue development, reducing parallel efforts.

Reshaping the policies, practices and regulations for Defense software acquisition may be another way of encouraging the rapid development, spread, and use of SEEs. Objections to this alternative are that an imbalance between the acquisition of hardware and software would result, that the Government cannot enter into commercial-like agreements, and that the Government needs to retain rights in data to development software in order to avoid being tied to a sole-source supplier of software and services for the life of the system. Responses to these objections are that imbalances abound in the current climate; the Government can enter into commercial agreements; and creative acquisition strategies can overcome proprietary barriers.

Modifying the rights in data clauses in acquisition regulations could permit contracting officers to enter into licensing

agreements with software vendors, encouraging the private sector to develop SEEs for license by the Government. Developing standards to coordinate the distributed, independent activities of the tool building community could assure that the private sector will develop SEEs and tools that are compatible.

Creating an industrial consortium is another way of effecting the development of SEEs. Such an arrangement would allow industry to pool its resources in pursuit of a goal from which all could benefit. There are many possible approaches to this alternative, making characterization of it difficult. The technical risk is also unknown because the consortium alternative does not specify a technical approach.

**Review of Key Issues**

No alternative alone can address all of the near-, mid-, and long-term needs for an SEE. Therefore, a combination of approaches will be necessary. A combination should also be less risky than a single strategy. Some key issues for selecting this combination for the near-, mid-, and long-term include DoD's needs, the available technical opportunities and choices, and the means used to obtain the environment(s), and the costs.

The overall STARS goal is to improve productivity while achieving greater system reliability and adaptability (in the face of increasingly demanding requirements) through software development and in-service support processes that are more responsive, predictable, and cost-effective. STARS is concerned with improving both the product (e.g., latent defects per thousand lines of code) and the process (e.g., productivity). The automated software engineering environment plays a key role in STARS' plans as both an essential element in achieving the goals and also in inserting improved technology into the Defense software community. In order to have the desired impact, the environment(s) must be widely accepted and used in the community. Such acceptance and use can be encouraged by involvement in the process of defining and obtaining environment(s). SEEs must evolve or be replaced to meet changing needs, while continuing to support existing software. Software developed in an environment should be transportable to a potentially different support environment, since developers of DoD software are seldom responsible for maintenance.

SEE capabilities are needed now, and a near-term SEE produced at low cost and low risk could have a positive effect on future SEE funding. By leveraging Defense system program funds and industry funds, STARS could increase the impact of its own funds. It is important that the costs of duplication of effort and tool incompatibility be avoided. Developing standards is one way of ensuring compatibility. General issues pertaining to the evaluation and selection of SEEs for the

near-, mid-, and long-term include capabilities, technical risks, transition and evolution, costs, and means of development and support.

Adapting or enhancing off-the-shelf commercial environments, commercial operating systems, or Government-sponsored environments such as the ALS (Ada Language System) could be a starting point for near-term development of a SEE. Approaches following either a centralized or decentralized design control would necessarily be more modest than comparable mid-term efforts. The most that could probably be done under centralized design control would be to develop a common core or methodology-oriented SEE. Sponsoring commercial development (decentralized design control) in the near-term might yield common core SEEs with orientations limited to a methodology, application, or organization. Revising policies (including business practices) and establishing workproduct interfaces are important near-term actions independent of the options with which they may be combined. Once SEE directions are established, tools and partial prototypes that are being developed for the mid-term can be used early in the near-term. It may also be possible to organize contractor IR&D and DoD MCCR system efforts to make them compatible and non-duplicative. Specifying interfaces and propagating the CAIS implementation at this point will also foster compatibility.

Expectations for the mid-term SEE include support for the entire life cycle, a high level of integration for DoD standards, and the ability to support, at reasonable cost, programming languages needed for post-deployment support of existing systems. The mid-term provides enough time to analyze, design, construct, and deliver a SEE from first principles, incorporating the best of previous work.

The possibilities for separately constructing portions of the environment(s) are illuminated by a "layered" model that separates dependencies on project, organization, application area, methodology, and the underlying computing system from the common core. The importance of common core interfaces makes it critical to establish them before higher layers are attempted. In addition, the cost of the common core will be large enough to probably make it impractical to do multiple versions unless significant non-STARS investment can be attracted. Such investment, however, could alternately be encouraged in the other layers with less of an issue of duplication.

Previously proposed alternatives related to the mid-term include the JSSEE, Industrial Consortium, Sponsoring Commercial Development, and Application-Oriented alternatives. The amount of duplication of effort that DoD should fund and the amount of investment from other sources that can be encouraged/achieved are key to selecting an alternative for the mid-term.

The main concern with respect to the long-term is transi-

tion to radically different paradigms. At a minimum, compatibility at the workproduct interface level should allow translation of the relevant subset of workproducts from the mid- to long-term environment. Very desirable would be adequate mid-term extensibility to incorporate radical paradigms or upward compatibility from mid- to long-term environments. Development of radical long-term capabilities will require earlier STARS and prototyping. Areas for R&D and concern for extensibility may include logic programming, expert systems, functional programming, transformal programming, and high-level end-user "programming."

**Recommendations for STARS Decision Making**

Recommendations for STARS decision making suggest an exploration and decision process as well as a number of possible activities. It is not expected that the SJPO (STARS Joint Program Office) should necessarily perform all of the activities.

The one conclusion that stands out the most firmly from the review of key issues is the importance of establishing early workproduct interfaces. Regardless of the other decisions made, the workproduct interfaces will play a significant role for some time, and they must be of high quality and extensible. The STARS Program should pursue them vigorously.

> Recommendation 1: Pursue vigorously the specification of workproduct interfaces and their standardization.

The decision process for other aspects that are not so firm should start with a review of goals by STARS top management. These should be covered in the sequence of long-term, mid-term, and finally near-term. The general relative importance of the various goals in the different periods should be established.

This review of goals, of course, relates to STARS overall programmatic goals. Its main purposes are to make sure that any impact of future goals is reflected back to earlier periods and that the relative importance of goals is seriously considered.

> Recommendation 2: Establish a STARS top management understanding of possible goals and their importance in the long-, mid-, and near-terms.

Next, a preliminary decision should be made on the overall approach, primarily from a capability viewpoint. This should address such overall issues as:

- Amount of capability desired and in what timeframe
- Number of different thrusts involved
- Amount of parallelism
- Level of continuity
- Rough level of effort involved over time

- o Relationships to current efforts
- o Decisions that can/should be delayed.

Recommendation 3: An (or at most a few) overall approach(es) in terms of capabilities and technical efforts structured over time should be established before considering the near- and mid-terms separately.

In the near-term, the exact capabilities and approaches will depend on the relative importance given the various goals. However, the issue of the basis or bases from which to start must be decided. Currently, one particular important piece of data is missing--ALS performance and quality. A systematic assessment of other possible bases is also missing although relatively recent assessments exist that could form a beginning for such an assessment.

Recommendation 4: ALS quality, performance and suitability as a basis for enhancement with near-term capabilities should be evaluated as soon as possible.

Recommendation 5: A systematic assessment of off-the-shelf alternatives that might form bases for enhancement should be performed such that results are available when a conclusion is reached in the ALS evaluation.

It should also be noted that the use of multiple bases reduces the risk of subsequent discovery of serious problems with any one of them. ALS could, for example, be included in such a set, even if it were not initially entirely suitable.

Recommendation 6: Consider combination strategies that do not create total dependence on any one high-risk element/implementation.

The near- and mid-term efforts should be reviewed for possible double-duty tasks that could combine tasks common to both to provide near-term capabilities and to contribute to mid-term efforts. Prototyping and early availability of mid-term tools are areas to look for possibilities.

Recommendation 7: Consider tasks that can both provide near-term capability and contribute toward mid-term capabilities.

Once technical efforts are sketched out for the near-and mid-terms, decisions must be made on task performers and acquisition strategy. The process of making these decisions will undoubtedly cause some iterations through earlier more technically-oriented decisions and will depend on a larger number of factors. However, some general recommendations can be made.

Recommendation 8: Avoid becoming overly dependent for suc-

cess on any one organization.

Recommendation 9: Involve all three Services, DoD agencies and a number of Laboratories and industrial firms.

Recommendation 10: Obtain quality expertise in all the necessary areas for each task.

Recommendation 11: Insure that continuous visibility and incentives exist and that milestones are frequent enough to allow for recovery.

Recommendation 12: Emphasize compatibility among mid-term efforts.

Recommendation 13: Strive for leverage and encourage the development of a marketplace in compatible tools.

This last recommendation needs some considerable early activity to establish the true potential for leverage in industry. The few inquiries made as part of investigating the Sponsoring Commercial Development previously proposed alternative were not encouraging, but DoD needs to collect harder data on industrial readiness to participate with the Government in this area.

Recommendation 14: Collect firm, verified information on industrial readiness and potential for leverage before making final decisions on acquisition approaches.

Finally, three general recommendations are made on the decision process itself.

Recommendation 15: Prepare a schedule for the needed decisions and insure that information generation or collection and staff work is done before decision point.

Recommendation 16: Avoid making final commitments sooner than necessary.

For example, final commitment on mid-term acquisition strategy can probably be delayed at least until the first set of workproduct interfaces is specified, i.e., probably late 1985.

Recommendation 17: Take advantage of all of the existing knowledge in this area since a number of organizations have been active, and can make useful contributions to the decision.

Recommendation 18: Recognize that SEEs are complex objects containing a number of potential portions or features with varying maturity, technical risks, ranges of possible sophistication, dependencies, cost, and benefits and that these portions and features may require varying treatment.

## 1. Background

One of the central items in the Software Technology for Adaptable Reliable Systems (STARS) Program is the automated software engineering environment (SEE)--that is, a set of capabilities that support software development and in-service support. It is a key to the timely, economically feasible production of reliable and maintainable software to meet the Department of Defense's current and planned military requirements.

This study investigates programmatic alternatives and related technical issues for meeting the DoD needs for software engineering environments in the near-(1-3 years), mid-(4-10 years) and long-terms. Goals and constraints are discussed, previously proposed alternatives outlined, and analysis provided. A major finding of this study is that only a combination of approaches, incorporating technical policy-oriented strategies, can succeed in providing the software production capabilities* that will be needed for military systems in the near-, mid-, and long-terms.

---

*These capabilities are variously referred to as software engineering environments, toolsets, and factories. The major distinctions between these capabilities seems to concern the degree of automation and integration. For extended discussions of toolsets and environments, the reader can consult the book Software Tools by B. Kernighan and T. Plauger (Prentice-Hall, 1975) and the paper "A Software Engineering Environment for Weapon System Software" by H. Steubing (IEEE Transactions on Software Engineering, July, 1984). The term software factory has recently gained currency as a means to identify automated software development and evolution environments whose "production lines" can be tailored for the unique mission critical software needs of individual systems (see also "A Computer-Aided Software Engineering Foundation for Software Factories" by J. Manley in the Proceedings of COMPCON, Fall 1984).

The study reported in (1) characterizes the current and future requirements of DoD software development. By virtually every measure of complexity, (e.g. size, cost) each succeeding generation of military technology requires mission critical software that is more complex than its predecessors. Increased operational requirements together with other requirements (e.g., extreme reliability) have led to software complexity that challenged the capabilities of the extant technology. Existing software production capabilities are often inadequate. A question left unanswered in the report is: Can future military requirements be met without taking special action?

The answer is almost certainly no. The demands of enhanced software requirements are growing too quickly. Consider, for example, software requirements for radar target acquisition systems. Early radar applications were not computerized. In the earliest embedded computer applications, only a few simultaneous targets could be tracked and identified. In current air defense systems - such as Aegis and Patriot - hundreds of targets must be tracked and engaged. Planning for space-based ballistic missile defense systems requires the coordinated tracking of tens of thousands of objects. Furthermore, this software must be essentially error-free, easily modified to adapt to changing threats and must use hardware resources very efficiently. Since existing technology can now barely keep pace with requirements, it seems unlikely that the same technology with normal progress over time can produce - within realistic economic limits - software whose basic characteristics exceed those of current systems by two or more orders of magnitude.

The risks of not taking action are very high. The danger in not responding to this technology shortfall is the potential failure to meet the mission requirements for future military systems of critical national importance.

This report concentrates on another option - applying special efforts to upgrade and modernize DoD software engineering capabilities via an automated software engineering environment or automated software "factory".

## 1.1 Goals and Constraints

A viable alternative for providing these capabilities to the Defense software community must satisfy two sets of constraints. First, the alternative must provide for the production of software that is suited to military mission critical computer resources (MCCR). Second, the alternative must be adaptable to the nature of DoD and Service system development, acquisition, and support.

The peculiar requirements of MCCR software are largely technical. They may be characterized by the simultaneous imposition of extreme system characteristics and parameters. This should be contrasted with non-MCCR applications, which may exhibit any of these characteristics but which rarely possess them in combination or in such extreme degrees. Typical of these characteristics are the following.

1. **Real Time:** A frequent requirement for MCCR applications is rapid real-time response. MCCR systems may have real time response requirements in the 10 microsecond range.
2. **Cost of Failure:** It is in the nature of military applications that system failures are catastrophic--failure costs are measured in terms of mission success or human life.
3. **Size:** The scale of MCCR application computer systems ranges from very small (e.g., the microprocessors of

Maverick) to very large (e.g., the large mainframe computers of WWMCCS (World-Wide Military Command and Control System)).

4. **Duration:** The development life cycle of MCCR systems tends to be very long. It is not unusual for system life cycles to span 15-20 years from inception to deployment. Once fielded many systems have projected life spans of 20 years or more.

5. **Maintenance and Support:** MCCR systems are subject to extreme support requirements. Systems may be subject to frequent modification. Availability requirements lead to very short time-to-repair characteristics. To complicate matters, maintenance personnel are rarely those who designed and constructed the system. MCCR software systems may be supported at sites far removed from their operational environment.

6. **Multiplicative Failure Rates:** In many applications, relatively low failure rates can be achieved by observing individual systems in isolation. By comparison, MCCR systems are sometimes duplicated in hundreds (e.g. aircraft) or thousands (e.g. missiles) of platforms, so that the number of instruction executions of MCCR software is very large. As a consequence, even very low failure rates can result in significantly many system failures.

7. **Controlled Development:** While many modern software development organizations impose programming standards and requirements, the DoD acquisition process results in developments that are controlled, monitored, regulated and constrained in ways that might be excessive in other applications.

These constraints have tended to limit the types of environments that are acceptable to the MCCR software development communities. For example, a recent Navy study of requirements for a software engineering environment (2) concluded with the following recommendations.

1. The software engineering environment must support the entire software life cycle.
2. The software engineering environment must be methodology driven.
3. The environment should support identified tools and techniques in each phase of the life cycle.
4. The environment should be capable of aiding in the configuration management of baselined software components that persist over the life cycle of the (target) MCCR system.

Each of the Services has studied the problem and derived sets of requirements on software engineering environments that respond to similar recommendations. The Navy recommendations, for example, were translated into a number of "guiding principles":

1. The software engineering environment must be consistent with the APSE (Ada Programming Support Environment). It must include full support for Ada and an integrated database. It must also provide support for existing standard languages.
2. The entire lifecycle of software is included in the software development process. This process and its related methodologies are the foundation of the environment.

3. The software engineering environment must support, to the greatest extent possible, the re-use of design components and software to eliminate re-invention.

4. The environment must be flexible enough to evolve, support new standards, and to support the re-engineering of software as it is upgraded.

5. The environment must accommodate a variety of host computers and computer configurations.

A viable alternative must promote the rapid, widespread use of the environment(s) that yield substantial improvements in productivity and reliability while meeting requirements such as those listed above. In evaluating alternatives, key parameters will include feasibility, costs, capabilities and benefits, risks, and timing.

Particularly important to the problem of timing are both providing significant capabilities in the near-term and avoiding a long, expensive software engineering environment development resulting in an environment that is not up-to-date when finally fielded. As yet, no clearly superior single approach to meeting Service requirements such as these has appeared. In fact, Government sponsored near-term approaches (e.g., contractural efforts to define and develop MAPSEs (Minimal Ada Programming Support Environments)) have met with schedule and funding delays that further limit the near-term options available to Service planning offices and software specialists in Program Offices.

## 1.2 Characteristics of Set of Previously Proposed Options

This chapter outlines and characterizes seven previously proposed alternatives for supplying near-, mid-, and long-term capabilities. Each option has been described by a vector of ten characteristics. (Longer descriptions are in the Appendix.)

**Time to Deliver:** Availability within three years is defined to be near-term, mid-term availability is in the four to ten year range, while long-term solutions are those that require more than ten years to deliver.

**Technical Risk:** Risk is an assessment of the probability of failing to meet objectives. Since exact risk assessments are notoriously difficult to derive for software intensive issues, risk will be characterized by the novelty of the technology required. For example, incremental improvements on demonstrated technologies are generally viewed as low risk. On the other hand, radically new technical approaches represent relatively higher risk. Different applications and technologies within an environment may have varying technical risks, making overall risk assessment complex.

**Design Methods:** Methodologies can vary between those that integrate and combine existing capabilities--the "bottom-up" approach--and those that require a "from scratch" design that may set its own requirements and translate into capabilities that are significantly different from those that are currently available (the top-down approach). While these extremes may affect technical risk, they are also important factors in determining risk associated with schedules and budgets.

**Funding:** Various sources of funding for implementing an alternative method of supplying environments are possible and range from Government-funded developments to those that are supported entirely with private sector funds. Intermediate sources include using Government funds to leverage an existing commercial investment and joint venture arrangements between Government and private sectors in which both sets of investors share overall risks and benefits.

**Implementors:** Qualified technical personnel in the private sector, Government/military, or intermediate organizations such

as the proposed Software Engineering Institute (SEI) (3) may design, construct or maintain an environment. "Commercial" refers to private sector ventures that are not Government-sponsored, whereas "Contractor" refers to a Government-contracted venture.

**Ownership:** Ultimate property rights in an environment may be vested in private or public hands. Technologies may be proprietary, or Government-owned.

**Business Model:** Alternative environments may be developed using Government acquisition and procurement practices, commercial leasing/licensing practices, or a combination of both. More than one business model may be appropriate for an option depending on its implementation.

**MCCR Acquisition Strategy:** Ownership, business models and risk assessments may suggest one of several approaches to placing an environment into the acquisition process when it is used for an MCCR application. For example, Government-Furnished Equipment (GFE) may be the appropriate approach in some situations whereas in others, it may be sufficient to specify interfaces and allow the MCCR developer to acquire the environment of his choice through purchase or license. In still other cases, it may be appropriate to employ current practices.

**Maintainer:** The options for support/maintenance personnel are the developer, a separately identified life cycle support contractor, or a Government software support facility.

**Management Approach:** The management of an environment development may be centralized, distributed to a large number of sites or decentralized to a small number of sites.

## 1.3 A Set of Previously Proposed Options

The options listed below have been proposed at various times over several years and each has advocates and critics ranging from various industrial and trade associations to senior

program management in the Department of Defense. This chapter does not take a position on the absolute desirability of adopting any single option, but rather seeks only to compare the alternatives relative to the ten characteristics identified in section 1.2 of this report. These are not the only characteristics of interest, however. Since we have not aimed for a comparative evaluation of alternatives, we have not attempted an exhaustive analysis of characteristics. Rather, we have concentrated on the characteristics which have been mentioned in recent reports, panels and critiques as having visible programmatic impact on STARS. A careful setting of priorities for these options should certainly consider other factors.*

### 1.3.1 The Joint Service Software Engineering Environment.

This Government-funded, Government-owned approach (called the JSSEE) has been launched by STARS and currently involves a number of DoD and Service Laboratories. The goal of the JSSEE is to produce a common software engineering environment to meet the needs of a wide range of DoD organizations, projects, and applications. The environment would be built on top of the Common APSE Interface Set (CAIS) virtual operating system and utilize previous MAPSE efforts. It would follow a life cycle oriented methodology such as defined in DOD-STD-SDS. Current approaches to JSSEE involve a top-down design of an environment starting with a common core capability.

---

*Other characteristics of interest include the technical aspects of the architecture, relationship to a methodology, extensibility, portability, existence of prototypes, resource utilization, and cost.

Some objections to the JSSEE revolve around a general concern that the top-down approach will not afford the environment the flexibility or extensibility necessary for the development of large scale software systems destined to be used in new and not always predictable ways. Critics argue that the requirements cannot be fully defined at the beginning of the environment development and that fixing requirements at the beginning creates an inflexible system that is not easily extended. Other objections are that the environment will be difficult to transport to new host machines or new development situations, that the JSSEE will take too long to develop, and that it will cost too much.

Advocates of the JSSEE believe that careful planning can overcome most of these objections. By using prototyping, they plan to gather feedback from users on requirements, solve design problems, and develop a flexible architecture to accommodate future needs. Environment integration, achieved by defining a set of interfaces, will result in an integrated collection of tools that will remain integrated as the collection expands and even if it supports an expanded set of methodologies.

Some issues confronting the JSSEE alternative are the need for an immediate capability, interoperability with tools developed for other methodologies or approaches to integration, the level of commonality and the technical feasibility of the JSSEE. A near-term option that might solve the need for an immediate capability would be for JSSEE to augment and distribute a Government-sponsored APSE such as the Army ALS (Ada Language System). Integrating the environment through a common set of interfaces will enable the JSSEE to accommodate tools developed for other methodologies or for other approaches to integration. The JSSEE plans to approach implementation of lower risk, better understood segments of the environment (e.g.

support for code and test) first while exploratory efforts proceed in higher risk segments (e.g. support for requirements analysis).

**1.3.2 Service Laboratories.** House and Senate Conference committees on 1984 Defense appropriations raised the following issue with respect to the STARS initiative: why can't the goals of the STARS program be accommodated within the structure of existing Service Laboratories? The management of a suitable software engineering environment in such a setting would be distributed with Government funding and ownership. Since the JSSEE team membership is drawn mainly from Service Laboratories, this alternative differs from the JSSEE alternative mainly in the choice of implementors, time to deliver, and technical risk. As opposed to the JSSEE alternative which is essentially mid- to long-term, Service Laboratories present a near- to mid-term and possibly lower risk alternative.

Laboratories can operate in an independent or cooperative mode to develop a software engineering environment. In either case, the functions of research (basic and applied), development and support, and communications and distribution, need to be assigned. One possibility is for the Software Engineering Institute to perform the research and software development and support functions and the Laboratories to assume responsibility for distribution, communication, and qualification functions.

For example, the Navy has adopted a model for Laboratory operation in which the developers of DoD software play a key role. The Laboratories qualify and demonstrate new technologies while external (usually academic) institutions have responsibility for research.

Objections raised to this alternative are that Laboratories have limited resources (both budgetary and personnel), and have not commonly developed advanced, modern software engineering environments in the past. In addition, using the Laboratory approach introduces the potential for costly duplication of resources unless Laboratories engage in extensive cooperation. Originally established to deal with physical and system-level engineering problems, Laboratories also may not be the best structure in which to develop and implement software.

However, creative Laboratory management can avoid some of these potential problems. Cooperative efforts between Laboratories that divide functions and pool resources are one way to partially deal with limited resources. Delegating some of the environment definition and development burden to the SEI (or its functional equivalent) is another solution to the problem of limited personnel resources. One advantage that Service Laboratories have over commercial settings is that Laboratory structures are flexible and not bound by market demands.

Laboratories may in some ways be a good near- to mid-term alternative. The incremental, evolutionary approach that personnel and budgetary limitations are likely to force Laboratories to adopt makes it a low risk option---but possibly with low payoff in the resulting capabilities as well.

**1.3.3 Off-the-Shelf Alternative.** An alternative that has a number of advocates is the adaptation and certification for military use of an existing commercial or Government-sponsored environment. This is a bottom-up near term strategy in which attractive off-the-shelf software development environments with

good track records are captured and augmented for DoD use. Examples of some available and soon to be available environments include Xerox Development Environment (XDE), Unix, ITT Programming Support Environment (PSE), and the Service-sponsored MAPSEs.

In discussing this alternative the method of adoption and various adaptation issues should be addressed. MCCR projects could adopt environments through license, purchase, or procurement. Most off-the-shelf environments will not be immediately suitable for MCCR use and will require adaptation. Some basic extensions to the core of the environment might involve rehosting, database management systems, human interfaces, and kernel software engineering environments. Another adaptability issue is whether an environment will be adaptable to multiple programming languages, such as Ada and other high-order languages. Methodology support is another adaptability factor, especially when there is an incompatibility between the methodology being imposed on the MCCR development and the methodology that is supported by the environment. The set of tasks supported by the tool set in the environment is another important consideration in matching an environment to an MCCR problem. It may be necessary to absorb MCCR-related tools as well as future tools.

Off-the-shelf environments are frequently dismissed as infeasible for the following reasons. The environments were not designed for MCCR applications and are therefore unsuitable. Proprietary barriers present long-term support risks of sole source reliance. A proliferation of unstandardized environments introduces new risks to the software development process. Industry will not make available for MCCR applications their most attractive environments for fear of losing their rights to the environment.

Many of these assertions are false. Although not designed for MCCR use, many commercial environments are being applied to applications that possess the extreme requirements of MCCR applications. Creative acquisition strategies can reduce the risks posed by proprietary barriers. Indeed, creative and well-managed software test and evaluation has been demonstrated to be a determining factor in the successful transition between development and maintenance environments.

1.3.4 Sponsoring Commercial Development. This is essentially the model upon which the DoD Very High Speed Integrated Circuit (VHSIC) program is based. The essence of this strategy is to identify and target a restricted number of industrial sites for further development. Criteria for enhanced Government investment in these sites might, for example, be based on the extent to which Government funding leverages an existing commercial investment in the environment. Once capabilities are enhanced at the selected sites, other technology transitioning programs would be implemented to ensure widespread access to the technology. This alternative may exhibit a mixture of characteristics depending on the existing capabilities at a selected contractor's site.

A number of factors are critical to assessing the overall preparedness of industry to participate in this way. One consideration is whether or not the company uses a software development approach or methodology that is suitable for automation in an environment. Another consideration is the degree to which the methodology is automated and whether the company is willing to adapt and export the automated capabilities. Success of this approach depends on a large total industry and Government investment.

The basic strategy for this alternative is to conduct an assessment to identify candidate environments and methodologies for investment by the Government. A small number of candidate environments would be selected as a result of the assessment. Selection criteria could include the cost of development yet to be undertaken, the extent to which the capability represented advances the state of practice for software development, the degree of technical risk involved in completing the environment, the willingness of the company to transition its technology, and the global assessment of how MCCR software development would be affected by the availability of the environment over near-, mid-, and long-term horizons. Contracts would be awarded for the completion and delivery of environments and tool sets that support the selected methodologies. The final task would be to successfully transition the developed environments into use.

Three primary objections can be raised to this alternative. The first is that the approach is based on a false comparison between software and integrated circuits. The second is that it throws Government support behind a few contractors that will eventually result in favoritism. Third, industry has not prepared itself and is seeking a Government subsidy for Research and Development (R&D) programs that should be supported privately.

Indeed there are important differences between software and integrated circuits. It is difficult to measure productivity improvements and return on investment for a software engineering environment. The essential outcomes of the selection and contracting process will have to be easily identifiable technological innovations. Not only are such innovations rare, but they are not always identified when they first appear.

The second objection is difficult to counter; it does seem that this alternative might give some contractors an unfair advantage, ultimately limiting competition and channeling technology into a few relatively narrow paths.

Inquiries of a small cross-section of seven industrial organizations gives some indication of industrial preparedness, the issue raised by the third objection. The organizations were questioned about three aspects of their environment capabilities: (1) methodology, (2) automated environments, and (3) the extent to which Government investment would be desirable. Six of the seven organizations claimed to use a methodology for software development; three cited the Yourdon methodology and the remainder actually only had plans to develop a methodology or confused it with the life cycle model in effect in their organization. No group reported documents or experience with a life cycle methodology. Only one of the seven claimed any degree of automated support for their methodology and this turned out to consist of a user interface capability. None of the organizations queried had a history of or plans for creating an integrated environment architecture. All of the groups expressed interest in Government support for tool development although they wanted to retain rights to the products developed, claiming that the Government investment would be in "research" and not in the product.

**1.3.5 Application-Oriented Environments.** This alternative is compatible with any of the alternative commercial options and has been advocated by at least one large trade association. The key concept is to support the development of environments with features and capabilities that are tailored to specific applications. It is based on the assumption that software production for specific applications, military avionics, for

example, requires environments and methodologies that are significantly different from those arising in another applications such as command and control.

The major objection to this approach is the lack of commonality among environments, making integration into a single environment or a small set of alternative environments difficult or impossible. Environments might also include proprietary information that is unavailable to other elements of the DoD community. The environments would attend only to the problems of developing and supporting software, not to the problems of acquiring software. This alternative could result in a high degree of duplication of effort that may not be justified in terms of the risk involved. Finally, software engineering environments that are not methodology and life cycle driven may fail to be general enough to gain widespread use, resulting in a proliferation of environments.

The problems of ownership and availability of the produced environments have to be resolved by innovative approaches to government/industry rights in data issues. The other objections can be addressed by developing a plan that recognizes and tries to minimize potential problems.

Such a plan might resolve the problems of duplication of effort and the inability to integrate the produced environments by developing environments in phases, gradually reducing the amount of parallel effort. Figure 1-1 illustrates the logical organization of the environment to foster commonality. Several application-oriented environment development projects could be started in parallel (Phase I). A public review at the end of Phase I would identify potential commonality among parts and decide which of the parallel efforts should carry on the development of common parts. Phase II would result in a design for the environment and brief plans for implementing, demonstrating, and transitioning the environment into use. Another public review would be held to scrub and homogenize the designs and to decide which efforts would proceed further.

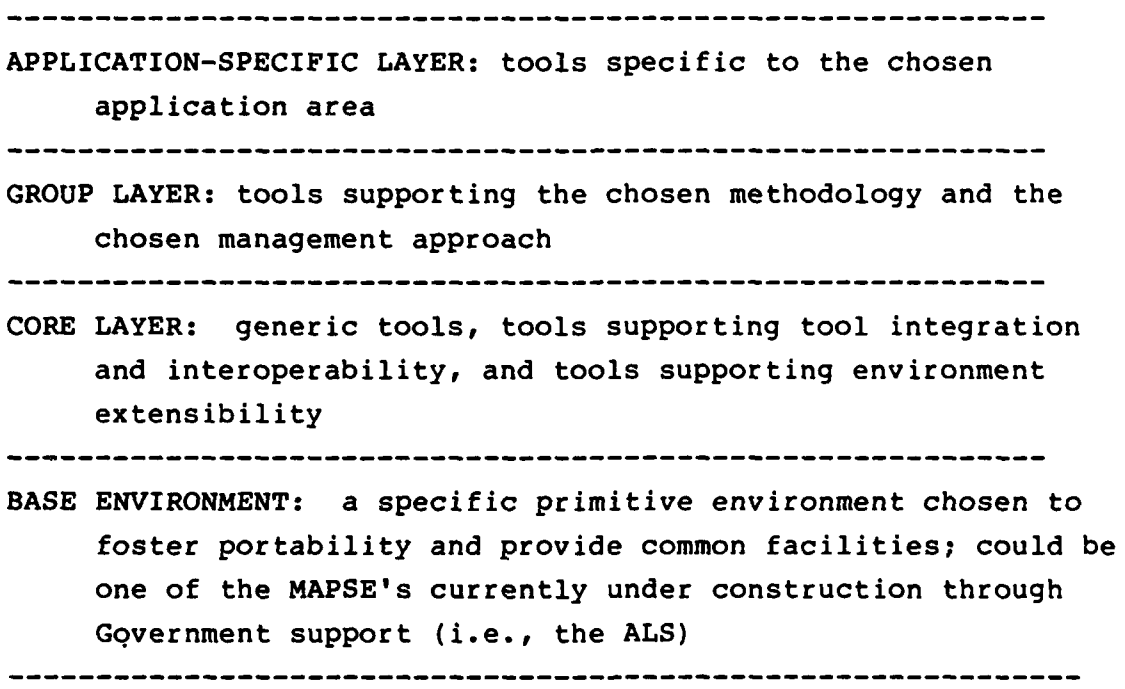



**Figure 1-1: An Organization for the Multiple Development Efforts**

Phases III and IV would proceed to develop and demonstrate the specific application-oriented environments utilizing the commonly built parts. While only a skeletal plan, this plan would provide a basic approach to developing application-oriented environments, reducing the duplication of effort and increasing the probability that the resulting environments could be integrated.

**1.3.6 Revising Policy.** This alternative involves reshaping the policies, practices and regulations in Defense software acquisition that may be inhibitors to the rapid spread and use of advanced software production techniques. Strictly speaking, this alternative is a variation of existing strategies. Policy approaches appear to be widely favored in industrial and in some DoD circles. For example, a STARS-sponsored "Technical Working Group" convened by the Institute for Defense Analyses to make recommendations concerning Rights in Data Issues as they apply to software development technologies advocated a radical shift in the form and content of acquisition regulations in this area.

There are some objections to this approach. Some critics are concerned that changes in the acquisition process for software would create an imbalance in the acquisition of hardware and software. It can be argued that imbalances abound in the current climate and that an imbalance in the acquisition process is secondary to the goal of improving the development and engineering of operational software. It is commonly, and incorrectly believed by many critics of this option that as a matter of policy, the Government cannot enter into commercial agreements such as licensing agreements. Critics also argue that the Government needs rights in data to development software in order to insure the maintainability of the delivered software and to avoid being tied to a sole-source supplier of software and services for the life of the delivered system. As stated elsewhere, creative acquisition strategies can overcome

proprietary barriers. There is some concern that modifying acquisition policies too drastically introduces new elements of risk that affect the stability of the acquisition. The response to this objection is that evolutionary forces by themselves are not sufficient to create the technology required to build the software in the next generation of DoD systems.

As part of this option, as well as others, standards could be developed to coordinate the distributed, independent activities of the tool building community. Standards could be defined for development, the methodologies to be used, and the interfaces among the tools that support these methodologies. Then, a mechanism would be developed for certifying tools and making them available for use in an environment. The final step would be to evolve the standards to reflect new software technology and upgrade the collection of certified tools.

Several objections to this approach include unnecessary duplication of effort resulting from a lack of more direct coordination, the difficulties in producing and certifying tools, and the potential that standards will either be too strict, leading to few tools achieving certification, or too lenient, leading to a lack of integration in the tool collection. It is possible to construct a plan to minimize the negative effect of various problems. One possibility is to create pools of tools. Pools could be application-specific or local to a project. Before entering the pool, a tool would have to be certified. An initial selection filter would classify tools according to a taxonomy, making it easier for potential users to identify tools of value to their project. Pools would consist of applicable, existing tools, and tools developed to make the overall collection complete. As tools are developed or re-engineered they would go through the certification process before entering the pool.

The major barriers to effective modification of acquisition practices are the rights in data clauses in acquisition regulations. These regulations can be modified to permit contracting officers to enter into licensing agreements with software vendors. Such a license would allow the contractor to maintain most rights to the software even if the software were developed in part at Government expense. The Government would have the right to use the software solely for its own purposes. If the software were developed solely at the contractor's expense, the developer would receive a negotiated royalty for use. The Government could direct the software developer and an inquiring party to enter into a direct licensing agreement rather than disclose proprietary information. The terms of the license should include steps that the contractor can take to protect proprietary information and offer the possibility of seeking damages if a license is breached.

1.3.7 Industrial Consortium. The consortium model is a variation of the commercial alternative inspired by the appearance of organizations such as the Microelectronics and Computer Technology Corporation (MCC) and the Semiconductor Research Corporation (SRC). In its simplest form, the consortium can be the private sector analog of the JSSEE approach. The essential characteristics of the option are the pooling of commercial resources to specify and design--in top-down fashion--a suitable environment. Ownership alternatives, impact on acquisition strategies and management options vary depending on the details. It is possible that the proposed SEI (Software Engineering Institute) can play a role in interfacing with an industrial consortium.

Organizational, legal, financial, technical, and marketing issues must be resolved before this alternative could be implemented effectively. Several alternatives are possible for

each. Consortia can be staffed by a variety of combinations of academic and industry personnel in addition to new hires. One staffing-related objection to the consortium approach is that consortia tend not to attract experienced industry people. "Borrowing" staff from industry for extended time periods solves that problem. A legal objection to consortia is that they violate antitrust laws. The Justice Department allowed the MCC to be organized as a for-profit corporation by its shareholder companies and is reviewing several other similar proposals. However, MCC is allowed to only do pre-competitive R&D not produce products.

Financial issues in setting up a consortium include finding a favorable tax climate, seeking land and/or equipment donations to get the consortium going, and handling joint funding. Technical issues include handling proprietary rights and the rights in data issue, selecting a technical approach to building a software development environment, and assigning responsibility for the resulting product. It is also necessary to define the market for which the environment is to be developed.

## 1.4 Inadequacy of Previously Proposed Alternatives

A summary of the characteristics exhibited by the seven options listed above is given in Figure 1-2. Since we have not assigned weights to the characteristics or utilities to their values, no clearly preferred choice emerges from this table. The Appendix describes each alternative in greater detail, discusses objections, and responds to the objections.

The picture that does emerge from Figure 1-2 is that a "single strategy" approach is very risky. If the STARS program is to address near-term options that offer immediate improvements in capability, mid-term options that reflect current assessments of military need (1) and long-term options that can incorporate technology advances that are as yet

| | CHARACTERISTIC | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| OPTION | TIME TO DELIVER | TECHNICAL RISK | DESIGN METHOD | FUNDING | IMPLEMENTOR | OWNERSHIP | BUSINESS MODEL | MCCR ACQUISITION STRATEGY | MAINTAINER | MANAGEMENT APPROACH |
| JSSEE | Mid-to long-term | High | Top-Down | Government | SEI or Contractors | Government | Government | GFE | SEI and Services | Centralized |
| Service Laboratories | Near-term to mid-term | Low | Top-Down | Government | Government or Contractors | Government | Government | GFE | DoD Labs or Contractors | Distributed |
| Off-the-Shelf Alternative | Near-term to Mid-term | Low | Bottom-Up | Government to Private | Contractors to Commercial | Government or Proprietary | Government or Commercial | Specify Interfaces | Developer | De-Centralized |
| Sponsoring Commercial Development | Near-term | Low | Bottom-Up | Government | Commercial | Proprietary | Government | Specify Interfaces | Developer | De-Centralized |
| Application-Oriented Environment | Near-term to mid-term | Low | Bottom-Up | Government to Private | Contractors | Proprietary | Commercial | Specify Interfaces | Developer | Distributed |
| Revising Policy | Near-term | N/A | N/A | N/A | Commercial | Proprietary | Commercial | Specify Interfaces | Developer | Distributed |
| Industrial Consortium | Mid-to long-term | Low-High | Top-Down | Private or mixed | Commercial or mixed | Proprietary | Commercial | Unknown | Developer or SEI | Centralized |

Figure 1-2.

unforeseen, a combination of approaches is essential since none of the options shown in Figure 1-2 address all of the near-, mid-, and long-term issues.

Another reason for favoring a combined strategy is the peculiar mix of options studied here. Some options address the technology itself (e.g. JSSEE or the Off-the-Shelf Alternatives) while others concern how the technology is to be developed and inserted (e.g. Service Laboratories and Industrial Consortium). Therefore, a feasible strategy for delivering software engineering environments will inevitably be a combination of these alternatives (the technology and the way in which it is developed and inserted).

## 1.5 Contents of Report

Given the inadequacy of any one of the previously proposed alternatives, Chapter 2 explores the possibilities of combining portions of them, looking first at the near-term, then at the mid-term, and finally at the long-term. The results of Chapter 2 lead to recommendations given in Chapter 3. The Appendix contains longer descriptions of the previously proposed alternatives, discusses more fully some points made in the body of the report, and reviews other arguments.

## 2. Review of Key Issues

This chapter reviews the key issues for STARS decision making -- first in general and then for the near-, mid-, and long-term possibilities. These issues include DoD's needs, the technical opportunities and choices available (including resulting capabilities and combining approaches), the means used to obtain the environment(s), and the costs involved.

### 2.1 General Needs

The overall STARS goal is to improve productivity while achieving greater system reliability and adaptability (in the face of increasingly demanding requirements) through software development and in-service support processes that are more responsive, predictable, and cost-effective. STARS is concerned with improving both the product (e.g., latent defects per thousand lines of code) and the process (e.g., productivity). The automated software engineering environment plays a key role in STARS' plans as both an essential element in achieving the goals and also in inserting improved technology into the Defense software community. In order to have the desired impact, the environment(s) must be widely accepted and used in the community. Such acceptance and use can be encouraged by involvement in the process of defining and obtaining environment(s).

Since MCCR requirements and SEE technological opportunities will expand over time, the environment(s) in use must improve either through evolution or replacement. However, existing software must be supported even if it is incompatible or insufficient in relation to the new SEE technology.

In-service or post-deployment software support is often done by the Government rather than the original development contractor. This fact together with the need to avoid contractor lock-in means it must be possible to deliver the

software (including all its specifications, designs, tests, etc.) in a form that can be supported by the DoD's environment(s)--or in some cases another contractor's environment.

In obtaining environment(s) for MCCR projects, STARS would, of course, like to obtain capabilities early, at low cost in STARS funds, and at low technical risk. Having some capabilities available in the near-term might also have a positive effect on out-year funding decisions for the STARS Program.

The impact of STARS funds can be increased if they leverage (cause to have spent) other funds, particularly Defense systems program funds and industry funds. In order for this to be most effective, the results of these investments must be cumulative and widely available in the Defense community. Duplication that is unwarranted by risk must be avoided, and the capabilities developed must be technically compatible and not restricted to the developer. In order to avoid the necessity for a significant amount of coordination, the ability to independently develop compatible tools would be very desirable.

The most obvious way to ensure such compatibility is to standardize interfaces within and between environments (13). The most attractive candidates for standardization are information contents, formats, and notations for deliverable work products in the SEE database. Defining the work products in each phase of the life cycle automatically defines interfaces for tools that input or output them (2). The new Joint Logistics Commanders' standards provide part of the needed tri-Service basis for definition and standardization. Standardization of work products also addresses the need for handover for post-deployment support, and avoiding contractor lock-in.

Different interfaces might also be considered for standardization including other kinds of data, invocation

interfaces, and external interfaces. Some level of standardization for compatibility is needed. This is also consistent with the DoD's statements to Congress on plans for computer systems interface standardization.

## 2.2 General Issues

A number of issues are briefly discussed to set the stage for the near-, mid-, and long-term as they recur throughout. These include technical capabilities, technical risk, transition between periods and approaches, cost, and means of development and support.

### 2.2.1 Capabilities of SEEs

Large variations in capabilities are possible among entities all of which are labeled as software engineering environments. In addition, capabilities will vary for a single environment as enhancements are made to the environment over a series of releases.

In discussing capabilities, this section concentrates on the types of activities supported and the types of tools and features that are in the SEE rather than the quality of the resulting software development process for MCCR software. Comparing what is included in different SEEs along with their maturity, performance and reliability (e.g., production quality versus research quality) is more easily done and reveals most of what is needed in this decision making context.

The spectrum of the life cycle covered by SEEs varies, although many today cover only the coding and testing phases. Important considerations are whether the following are supported:

- o word/document preparation as well as graphics capability
- o management tools and the ability to handle large projects

- o specific programming language for specific target machine
- o support of or compatibility with relevant DoD standards
- o application-oriented, methodology-oriented or organization-specific capabilities

In addition, the sophistication of the human interface, the commands available and the level of integration vary widely and are important considerations for ease of use.

Other features of interest include the availability of the SEE on different computers and operating sytems as well as its conformance to relevant standards--e.g., Ada, CAIS. Particularly in the long-term, the underlying technical basis may be of interest, for example, whether it has an expert systems or functional programming underpinning.

A spectrum of environment capabilities can exist and is a key consideration in the decision making process.

### 2.2.2 Technical Risk

Of course, technical risk varies with the type of capability or feature and its sophistication. It also increases as the lifetime projected for an environment increases or the level of integration attempted increases since the incorporation of new, unanticipated technologies becomes more and more uncertain. Lastly, risk exists in any SEE development effort; proven environments involve less risk, although potentially lower payoff.

Technical risk is not a simple issue. It can differ significantly from one segment of an environment to another. It generally is higher the more ambitious the effort. One must be careful to avoid choosing an option with lower risk but with even more significantly lower capability.

### 2.2.3 Transition and Evolution

Requiring projects or people to move from one environment to another is a step that must be seriously considered. Transition strategies and migration aids are generally necessary. The requirement of at least one move from the various current operating systems and tools in use to a new SEE may be unavoidable for those who desire to take full advantage of the new capabilities. Even in that situation, a good transition strategy and good training will be very useful for minimizing the change and disruption. Any plan for several radical changes over the coming ten years and beyond will incur substantial costs that must be weighed in decision making.

Evolutionary strategies, for example, the use of an architecture designed to facilitate change, represent an approach to easing transition. Another strategy is to encourage upwards compatibility where the new environment subsumes the old.

The ability to learn from the experience with the existing environment in order to provide appropriate future enhancements is a consideration. Ease of prototyping future enhancements is an issue too, but less of a requirement since special prototyping efforts can be done if necessary.

The ease of movement and adaptation of new advances by projects, organizations, and persons is important. If the environments involved are largely incompatible and substantial changes are required, the cost is significant. The adverse impact of "change" must be traded-off against the advantages the changes bring.

### 2.2.4 Costs

The desirability of leveraging industry or commercial investment to reduce costs to STARS has already been discussed. Three other issues related to cost are relevant. First, of

course, is the fact that more capabilities generally cost more and cost and benefit both must be considered. Duplication of effort or incompatibility caused by multiple implementations need serious consideration and strong justification because the additional cost can be significant. Second, true cost avoidance will sometimes be possible by using or adapting existing products--Ada compilers and MAPSEs are good possibilities. Finally, payment or repayment schemes through royalties or other arrangements can potentially relieve the government investment burden and are probably a necessity if an active marketplace in tools is to be created.

### 2.2.5 Means of Development and Support

Potentially a number of different types of organizations could be involved in the SEE development and support, including:

- DoD Software Engineering Institute (SEI)
- Service Laboratories
- Defense systems contractors
- Defense software contractors
- Commercial software houses
- Computer manufacturers
- Small entrepreneurial specialty firms
- Universities.

The SEI is expected to be involved in a technology insertion role although different programmatic and technical choices could cause variations in that role. A spectrum of development approaches is possible. A single environment as a whole could be done in one place; it could be split up and portions done at different places with an integration organization (say, SEI); it could be done by a prime contractor with substantial subcontracting; or it could at least in part be allowed to grow organically with portions contributed by various organizations. Multiple environments could be developed by

different organizations with competitive runoffs of architecture, designs, prototypes, or even environments. It should be noted that involvement of an organization in development can aid in its acceptance and use of results. This would be a side benefit resulting from this involvement.

One key issue for the means of development and support is design control, i.e., whether the design, interfaces, and changes will be under DoD control. In preparing the SEI management plan, the issue of DoD design control was considered very important and played a key role in the rationale for establishing a Configuration Control Board over the SEI even though the SEI would already have close coordination with DoD. An important consideration is the extent to which design control should be decentralized--e.g., control of interface standards versus implementation details, or control of the environment core versus methodology or application-oriented tool sets.

## 2.3 Near-Term

This section covers the near-term issues of DoD needs and desires, possible approaches, possible performers, and rescue/acceleration of efforts. Although many of these can also be considered as issues for the mid-term and long-term, they are particularly important issues in the near-term.

### 2.3.1 DoD Needs

Needs to be emphasized in the near term depend on the overall multi-period strategy chosen and STARS programmatic decisions. Greater or lesser emphasis could be placed on:

- Improving existing projects
- Getting new projects off to good starts
- Learning from experience
- Transitioning to future environment(s)

- o Facilitating the introduction and use of Ada
- o Showing results that will impress decision makers on future funding.

In prior STARS plans, the last three points have been emphasized.

### 2.3.2 Possible Near-Term Approaches

Numerous possibilities exist for near-term approaches and are listed in Figure 2-1. For the moment, this list does not consider who would perform the tasks. Many of the possible approaches in Figure 2-1 could be done in single or multiple implementations, or with competitive run-offs. The discussion in this section follows the ordered topics in Figure 2-1.

The Army's ALS can be considered off-the-shelf if it indeed makes its planned January 1985 date for a production-quality release. However, it has had a history of problems and neither the date nor the quality can be assumed. Nevertheless, it potentially provides a minimal environment that could be enhanced.

Several operating systems have or will shortly have Ada compilers available for them--Data General, Digital's VAX/VMS (TM), and some implementations of AT&T's UNIX (TM) or UNIX-look-alikes are the prime examples. Some of these might be used as the base. Interestingly, the Swedish effort at a near-term Ada environment will start on VAX/VMS and is planned to also be made available on UNIX.

Off-the-Shelf and Enhance/Adapt

- ALS
- Existing Operating Systems plus Ada
- Existing Commercial Environments (Adapted plus Ada)

Development with Centrally Controlled Design

- Common Core
- Methodology-Oriented

Sponsor Development

- Common Core
- Methodology-Oriented
- Application-Oriented
- Organization-Oriented

Revise Policy

- Business Practices
- Specify Work Product Interfaces

And After *Decision on Direction(s)*

- Early Availability of Some Mid-Term Tools
- Usable Prototypes
- Organize DoD Programs and Contractor IR&D to Contribute
- Specify Other Interfaces
- Propagate CAIS Implementations

**Figure 2-1: POSSIBLE NEAR TERM APPROACHES**

UNIX, with the tools available for it, is on the boundary between an operating system and an environment as the terms are used in Figure 2-1. Using Mesa, XDE, UNIX or another modern programming support environment as a starting point is a possibility--one that becomes even more attractive if ALS does not perform up to plans. Using much of the design of an existing environment--such as Mesa or its possible successor that is still in the R&D stage, Cedar--is also a possible way to accelerate a near-term development.

The development of a near-term software engineering environment (starting from an off-the-shelf basis) can essentially be approached in three ways from the viewpoint of design control:

- Centrally controlled design (e.g., a mini-JSSEE)
- Design controlled by developing organization (e.g., contractor in sponsoring commercial development)
- Specify interfaces (e.g., specify work product interfaces).

The specification of work product interfaces, as discussed earlier, has much merit and a consensus is emerging that work product interfaces are the appropriate level to specify--as reflected in the discussions at the Joint Logistics Commanders Orlando I Conference in the Fall of 1983.

The common core of a SEE is a substantial portion often including the underpinnings such as database management, human interface, command processing, office automation, common management tools,as well as most of the tools in the well understood areas of code and test. When centrally controlled design is discussed, it is usually in terms of this common core plus some other interfaces (such as work products) outside the code and test tools.

Methodology-oriented portions of a SEE, particularly requirements and design methodologies, are areas with substantial R&D activity and thus a variety of opinion about appropriate choices exist. STARS has an effort already launched by the Ada Joint Program Office and known familiarly as "Methodman" to help guide this area over the next several years. However, quite possibly, there will be no one "best" methodology or allegiances to them within the Defense community will all prove so strong that several requirements and design methodologies will need to have supporting tools within the SEE(s). Interestingly, the JSSEE planning team has made an initial estimate that approximately three such methodologies will eventually need to be supported.

In general, the issue of doing multiple versions of some portion of environment(s) involves questions of:

- o Risk, poorly understood or difficult to develop
- o Different needs, such as different application areas
- o Acceptance and use of environment
  - Involvement of more organizations in development
  - Strongly different desires that cannot be met by tailoring
  - Availability on different computers
- o Compatibility, whether tools and database contexts from one be used usefully with other tools from another
- o Cost.

If multiple efforts were launched today, there is little in place to help ensure any compatibility. Even the CAIS specifications planned for early 1985 will ensure little compatibility (only transportability of tool sets and data). Establishing the work product interfaces will provide necessary minimal compatibility. More compatibility could be provided by more interface specifications and a common architecture; without compatibility among STARS-supported efforts they cannot

be cumulative. Thus, some delay in launching multiple efforts while the infrastructure for inter-tool compatibility is put in place could provide benefits in the future.

Revising business practices and establishing work product interfaces are important considerations independent of the option(s) with which they may be combined. It should be noted that in addition to specifying or standardizing interfaces, the capability to validate or certify conformance with them is needed. The APSE Evaluation and Validation effort launched by the Ada Joint Program Office has begun considering many of the issues involved.

Once SEE directions are established, if an effort is launched for the mid-term (or even the long-term), the early availability of some tools or partial prototypes that can be easily made usable, is a possibility. The tool efforts of various DoD MCCR system programs and contractor IR&D efforts can also potentially be organized so that their results are compatible and non-duplicative. In addition, if a commitment is made to CAIS or other similar standards, support for their propagation may be worthwhile.

### 2.3.3 Task Performers

A key issue that must be considered in the near-term is who will carry out the SEE development task, for whatever set of technical options chosen. This is particularly important if STARS supports efforts without significant DoD design control. Quality organizations and appropriately skilled personnel are essential to developing the advanced software that the SEE would comprise. In fact, the recent experience with the DoD's MAPSE contracts points out the substantial care that must be exercised in such acquisitions.

Expertise needed for different portions of a SEE vary and a range of specialists is required in addition to key system

designers. These various areas of expertise are probably not obtainable totally today from one organization. Application- and organization-oriented aspects seem to particularly require the special knowledge of organizations that perform in a given application area or whose peculiar organizational practices are being supported. However, contractor-subcontractor team(s) or consortia could be formed to collectively bring together the needed expertise. On the other hand, DoD could chose to divide up the work appropriately and have separate organizations carry out different portions. For compatibility, this would require prior interface establishment and final integration efforts--roles envisioned in some planning as being done by the SEI (and its precursors). In any case, the appropriate high quality expertise must be brought to bear on the various portions as well as on the overall design and interfaces.

### 2.3.4 Effort Rescue/Acceleration

In the near-term an additional problem that may arise is STARS becoming the "rescuer" of some product that was to form part of a firm base for SEE efforts. Among the possibilities are ALS, CAIS, and even some Ada compiler effort to which one might commit without prior adequate evaluation to insure its suitability or performance. Of course, STARS plans should avoid such opportunities for inverse leverage--spending funds on tasks that would otherwise have been supported by the Services or industry.

### 2.3.5 Summary

Thus, as one would expect, many near-term issues are general issues, such as design control, technical approaches, task performers, and budget. The near-term has a special problem in deciding the bases from which to start, particularly in off-the-shelf capabilities. Of course, near-term decisions

will depend in part on plans for the mid- and long-term whose issues will be discussed in the next two sections.

## 2.4 Mid-Term

Many of the issues related to efforts providing capabilities in the mid-term are similar to those discussed under near-term. In this section, we will try to point out what is different about the mid-term. In addition, potential transition issues exist concerning the transition from near-term to mid-term, and mid-term to long-term.

### 2.4.1 DoD Needs

During the mid-term (4-10 years), the expectations for STARS SEE(s) are naturally greater than for the near-term efforts. They include support for the entire life cycle, a high level of integration, support for DoD standards, and the ability to add support at reasonable cost for programming languages needed for post-deployment support of existing systems. In addition, in order to readily make emerging artificial intelligence based tools available in the environment, it should support Common LISP, the new DARPA "standard". Late in the mid-term need will also exist for integration with hardware and systems oriented support tools.

### 2.4.2 Mid-Term Possibilities

In the mid-term, it is considerably more difficult to project in concrete detail the possibilities that would be available. The capabilities either can be provided on the base of capabilities emerging from the near-term or as a separate effort. During the three years of the near-term, it would be difficult to analyze, design, construct and deliver a SEE from first principles incorporating the best of previous work. In

the mid-term, an opportunity exists to do the job right, including explorations of and experimentations on key points of technical risk before design commitment. The substantially greater capabilities expected in the mid-term, of course, imply substantially larger development efforts.

Figure 2-2 lists the topics to be discussed in this section.

- o Layered development
- o Timing and Releases
- o Design Control
- o Organize other fund sources
- o Marketplace

**Figure 2-2: Some Mid-Term Aspects of Possibilities**

If different portions of the environment are to be constructed by different organizations, then standard workproduct interfaces could also help provide some interfaces among the portions of the environment. However, when one considers the special expertise needed for various portions and especially the number of warranted versions in these portions (e.g., different application areas), then other interfaces become important as well.

In Figure 1-1, some layers were suggested in the previously proposed application-oriented approach. Figure 2-3 revises that diagram slightly and incorporates other aspects to show various portions that might be developed by separate organizations. The top three "layers" all would have multiple implementations aimed at supporting different projects, organizations, and application areas. The project- and organization-dependent portions should

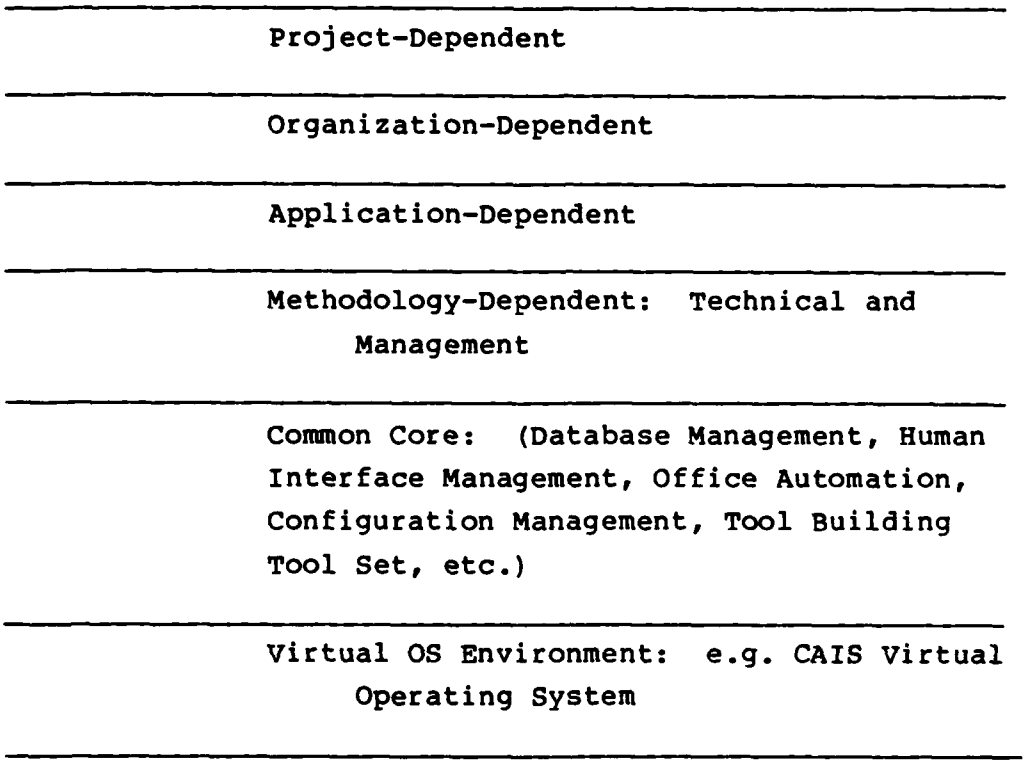



**Figure 2-3: Some "Layers" for Possible Separate Development**

be funded and controlled by the specific projects and organizations. They potentially would need to know the interfaces to the layers beneath, but certainly at least the common core level. The application-dependent layer would also have a number of different application areas, many of which could be developed somewhat separately. Application-oriented reusable software and tool sets would receive funding from STARS. In addition, a number of DoD organizations could reasonably be expected to want to support application-oriented efforts and would need to know the interface for the lower layers.

A dependence that cuts somewhat across organization, application, and project (although strongly related) is target computer dependence. STARS probably would fund at most one target-dependent tool set and that would only be if needed for demonstration of common integration/debugging capabilities in the SEE.

Several methodologies might have supporting tool sets developed by different organizations, either funded by STARS or by organizations with strong allegiance to particular methodologies.

The bottom layer is expected to have a number of implementations. This would certainly be the case if the CAIS virtual operating system approach were taken. Under certain scenarios, STARS would fund a few of these implementations for SEE development purposes. One example is to make SEE(s) available on workstations in a distributed environment.

The common core is the only portion in which it is reasonable to expect that a single implementation could suffice. (Risk or other reasons might justify some multiple implementation of subportions). Subportions of the common core could potentially be done separately if interfaces are established. In addition, the interfaces to the common services provided by this layer need to be known to the implementors of the layers above.

All these requirements for having known, established interfaces in the common core imply that efforts should be made to establish these interfaces early. Along these lines, the desire to avoid the costs of multiple implementations of the common core, caused the Services to recommend in 1983 that the application-dependent layer and common core not be bundled together. They recommended that the application layer be delayed until the common core layer effort had proceeded somewhat.

Thus, a number of opportunities exist to build portions of an environment separately, but this implies the need to specify interfaces at an early stage. In addition, a logical (partial) order exists in which to define the interfaces.

The other key issue that this discussion of layers and multiple implementations addresses is cost. The common core, even with maximum reuse of MAPSE efforts (and Navy extensions) probably has a development cost of $40-$100 million. Even with the low end estimate, multiple implementations would be difficult, given the projected STARS budget.

During the mid-term period, capabilities should be made available incrementally over a series of releases. If STARS has a lifetime of seven years, as currently envisioned, any capability should be available by year seven to be within the "STARS lifetime". On the other hand, a "do it right" effort probably cannot field much of an integrated release before year three or four.

When one uses the available software cost and schedule estimating models (as the JSSEE effort has), it soon becomes clear that to meet this window, significant parallelism is required. Even so, it is unrealistic to think that all parts of all layers could be developed by the end of this window. This reinforces the need for early definition of interfaces, and product evolution and extensibility.

The degree of design control is also an issue in the mid-term. As one goes upwards from the common core, design control becomes less and less of an issue. However, if multiple organizations are involved, the availability and stability of common core interfaces are very important.

In the mid-term, after key interfaces are established, other organizations besides STARS can invest with some hope of compatibility. DoD programs and Defense contractor IR&D could leverage the STARS investment. Undesired duplication in such efforts would be decreased by an information campaign combined with clearinghouse functions disseminating information on such efforts. Investment from commercial sources could also be encouraged in the hope that an active marketplace would be created in compatible tools.

### 2.4.3 Previously Proposed Alternatives

Previously proposed alternatives related to the mid-term include:

- Joint Service Software Engineering Environment
- Industrial Consortium
- Sponsoring Commercial Development
- Application-Oriented.

Industrial Consortium is really a means (like Service Laboratories) and does not currently imply an explicit technical choice.

Sponsoring Commercial Development and Application-Oriented are essentially near- or mid-term strategies. Off-the-shelf (including ALS) alternatives could be precursors to these or other mid-term alternatives.

Two key cost considerations influence mid-term choices--how much duplication of effort (in lower levels of Figure 2-3) should DoD fund and how much investment from other sources can

be encouraged/achieved. JSSEE might encourage substantial non-STARS or non-DoD investment through standardized interfaces and proper encouragement. However, Sponsoring Commercial Development and Application-Oriented clearly, more directly motivate the contractors involved to invest funds of their own. Industry investment, early insertion into sponsored organizations, and greater contact with real-world contractor requirements are pluses for Sponsoring Commercial Development and Application-oriented but are hard to quantify.

Application-oriented could build on top of a JSSEE effort supplying only the application-dependent layer. So two separate issues exist -- general approach, JSSEE versus Sponsored Commercial Development, and inclusion of application-oriented aspects.

### 2.4.4 Summary

The mid-term is the first period in which a "do it right" environment could have a full range of capabilities available. The possibilities for separately constructing portions of the environment(s) is illuminated by a "layered" model that separates dependencies on project, organization, application area, methodology, and the underlying computing system. The importance of common core interfaces makes it critical to establish them before higher layers are attempted. In addition, the cost of the common core will be large enough to probably make it impractical to do multiple versions unless significant non-STARS investment can be attracted to do this. Such investment, however, could alternately be encouraged in the other layers with less of an issue of duplication.

## 2.5 Long-Term

In the long term all alternatives have the issues of compatibility on cost of conversion of SEE(s) to radically

different paradigms. Adequate extensibility of the mid-term environment(s) or upward compatibility in new long-term environments(s) would be very desirable. However, compatibility at the workproduct interface level that allowed translation of the relevant subset of work products is the minimal requirement.

To achieve radical long-term capabilities, significant R&D and prototyping will, of course, be required in earlier periods. STARS plans have included doing some of this-- particularly prototyping.

While the long-term is difficult to see clearly, some likely technologies can be identified today. These include

- Logic programming
- Expert systems
- Functional Programming
- Transformational programming
- High-level end-user "programming".

For many of these it should be possible to address the extensibility/compatibility issue in definition and design of a mid-term environment.

Thus, the long-term is of interest mainly in terms of transition and R&D concerns.

## 3. Recommendations for STARS Decision Making

This chapter outlines an exploration and decision process that is recommended for the STARS program to follow over the next year. The recommendations contain a number of possible activities related to the decision making process, and it is not expected that the SJPO (STARS Joint Program Office) should necessarily perform all of them.

The recommendations are discussed in a sequence that starts with conclusions that do not depend on the choices made, discusses goals, decisions on general approaches, then the decisions on near- and mid-terms. Where potentially obtainable information is lacking, actions are suggested for obtaining it. And finally, general suggestions are made on obtaining good decisions.

The one conclusion that stands out the most firmly from the review of key issues is the importance of early establishment of workproduct interfaces. Regardless of the other decisions made, the workproduct interfaces will play a significant role for some time, and they must be of high quality and extensible. The STARS Program should pursue them vigorously.

> Recommendation 1: Pursue vigorously the specification of workproduct interfaces and their standardization.

The decision process for other aspects that are not so firm should start with a review of goals by STARS top management. These should be covered in the sequence of long-term, mid-term, and finally near-term. The general relative importance of the various goals in the different periods should be established.

This review of goals, of course, relates to STARS overall programmatic goals. Its main purposes are to make sure that any impact of future goals is reflected back to earlier periods and that the relative importance of goals is seriously considered.

In the near-term, relative emphasis is especially important to establish. As mentioned in Chapter 2, greater or lesser emphasis could be placed on

- Improving existing projects
- Getting new projects off to good starts
- Learning from experience
- Transitioning to future environment(s)
- Facilitating the introduction and use of Ada
- Showing results that will impress decision makers on future funding.

Examples of activities to emphasize include development of the near-term capability needed primarily as a demonstration to affect funding, a set of capabilities for particular Defense programs, or a transition step toward mid-term capabilities.

Figure 3-1 might stimulate some thoughts in this area. It is a list of goals and objectives in priority order established by National Security Industry Association (NSIA) Working Group, and suggested to the JSSEE effort. Many of the points derive from the appendix on goals in the initial JSSEE plan (12). It is interesting to note that this prioritization is probably significantly different from one that would be produced by DoD. Of course, any such simple ordering cannot adequately reflect the complex interrelationships among the goals and objectives.

Recommendation 2: Establish a STARS top management understanding of possible goals and their importance in the long-, mid-, and near-terms.

**Figure 3-1: NSIA WORKING GROUP PRIORITIZATION OF JSSEE GOALS (5 June 1984)**

1. Develop a common integrated SEE adaptable to Service and Project specific environments and methodologies.
2. Production quality SEE by mid FY88
3. Show an improvement in software productivity (at least a factor of 2 by 1990)
*4. Provide evaluation of JSSEE prototype SEE by mid FY86.
5. Identify standard information content interfaces for baselined products in database.
6. Provide an integrated set of tools to support at least one software development methodology.
7. Make JSSEE portable to multiple host computers.
8. Support the use of the new JLC standards (MIL-STD-SDS)
9. Gain wide acceptance of industry and academia.
*10. Provide for evaluation and feedback of early use for subsequent baselines.
11. Support CAIS.
*12. Support Program and Project management.
*13. Provide training tools.
14. Develop an architecture for the JSSEE and a method for evolution.
*15. Base the JSSEE on a single Service MAPSE.
16. Provide a means to incorporate technological advances.
*17. Support current Service system development practices (languages).
*18. Exploit distributed architecture/networking.
19. Provide standard user interface.
20. Transfer to SEI.
*21. Provide multi-level security.

*indicates a new goal or an item listed as a design issue in (12).

Next, a preliminary decision should be made on the overall approach, primarily from a SEE capability viewpoint. This should address such overall issues as:

- Amount of capability desired and in what timeframe
- Number of different thrusts involved
- Amount of parallelism
- Level of continuity
- Rough level of effort involved over time
- Relationships to current efforts
- Decisions that can/should be delayed.

The rough levels of effort implied by different arrangements of capabilities and thrusts over the years in the STARS lifetime then must be compared with estimates of reasonableness. However, efforts should not be entirely removed from future consideration unless there appears to be no hope of obtaining funds from any source.

A firm decision on one set of efforts structured in a certain way over time is not necessary at this point in the decision process, but the possibilities should have been considered and narrowed to only a few.

> Recommendation 3: An (or at most a few) overall approach(es) in terms of capabilities and technical efforts structured over time should be established before considering the near- and mid-terms separately.

In the near-term, the exact capabilities and approaches will depend on the relative importance given the various goals. However, the issue of the basis or bases from which to start

must be decided. Currently, one particular, important piece of data is missing--ALS performance and quality. A systematic assessment of other possible bases is also missing, although relatively recent assessments exist that could form a beginning for such an assessment.

> Recommendation 4: ALS quality, performance and suitability as a basis for enhancement with near-term capabilities should be evaluated as soon as possible.

The first production quality release of the ALS is scheduled for January 1985. While visibility into its condition can be obtained before then, it is probably unfair (and dangerous) to reach any final conclusion before then.

The STARS program must be prepared in case the evaluation of ALS reaches a negative conclusion on its suitability. This need, coupled with the need to explore the possibility of multiple bases that could include the near-term capabilities and to gather information useful for mid-term efforts leads to Recommendation 5.

> Recommendation 5: A systematic assessment of off-the-shelf alternatives that might form bases for enhancement should be performed such that results are available when a conclusion is reached in the ALS evaluation.

It should also be noted that the use of multiple bases reduces the risk of subsequent discovery of serious problems with any one of them. ALS could, therefore, be included in such a set, even if it were not initially entirely suitable.

> Recommendation 6: Consider combination strategies that do not create total dependence on any one high-risk element/ implementation.

This is also useful advice for the mid-term (and, of course long-term). Once the near-term is laid out, then the mid-term efforts should be structured in more detail.

The near- and mid-term efforts should be reviewed for possible double-duty tasks that could combine tasks common to both to provide near-term capabilities and to contribute to mid-term efforts. Prototyping and early availability of mid-term tools are areas to look for possibilities.

> Recommendation 7: Consider tasks that can both provide near-term capability and contribute toward mid-term capabilities.

Once technical efforts are sketched out for the near-and mid-terms, decisions must be made on task performers and acquisition strategy. The process of making these decisions will undoubtly cause some iterations through earlier more technically-oriented decisions.

The issues involving task performers, organizational involvement, leveraging funds, need for quality, etc. have been discussed at length. Decisions will depend on a larger number of factors. However, some general recommendations can be made.

> Recommendation 8: Avoid becoming overly dependent for success on any one organization.

> Recommendation 9: Involve all three Services, DoD agencies and a number of Laboratories and industrial firms.

> Recommendation 10: Obtain quality expertise in all the necessary areas for each task.

Recommendation 11: Insure that continuous visibility and incentives exist and that milestones are frequent enough to allow for recovery.

Recommendation 12: Emphasize compatibility among mid-term efforts.

Recommendation 13: Strive for leverage and encourage the development of a marketplace in compatible tools.

This last recommendation needs considerable early activity to establish the true potential for leverage in industry. The few inquiries made as part of investigating the Sponsoring Commercial Development previously proposed alternative were not encouraging, but DoD needs to collect harder data on industrial readiness to participate with the Government in this area. Currently, it appears that inquires should come directly from DoD and that site visits will be needed to understand and verify the information. In addition, specific definition of the possible capabilities in future SEEs are needed to insure that all have a common understanding about the same capabilities.

Recommendation 14: Collect firm, verified information on industrial readiness and potential for leverage before making final decisions on acquisition approaches.

Finally, three general recommendations are made on the decision process itself.

Recommendation 15: Prepare a schedule for the needed decisions and insure that information generation or collection and staff work is done before decision point.

Recommendation 16: Avoid making final commitments sooner than necessary.

For example, final commitment on mid-term acquisition strategy can probably be delayed at least until the first set of workproduct interfaces is specified, i.e. probably late 1985.

Recommendation 17: Take advantage of all of the existing knowledge in this area since a number of organizations have been active, and can make useful contributions to the decisions.

Recommendation 18: Recognize that SEEs are complex objects containing a number of potential portions or features with varying maturity, technical risks, ranges of possible sophistication, dependencies, cost, and benefits and that these portions and features may require varying treatment.

# APPENDIX

# Appendix

## 1. The Joint Service Software Engineering Environment

An effort to define and begin design for a Joint Service Software Engineering Environment (JSSEE) has been launched by STARS under a tri-Service team with Navy leadership. An initial plan and a draft operational concept document with high level user requirements have been prepared.

This section provides a description of the technology to be incorporated in the JSSEE, clarifies objections to the JSSEE approach and raises a number of related issues (such as the short-term delivery of APSE capabilities by augmenting a Government-sponsored environment such as the Army ALS, Navy ALS/N, or Air Force AIE (Ada Integrated Environment).

### 1.1 Description of Approach

To implement the JSSEE, an architecture and important interfaces must be established (see section 6.3 for rationale on interfaces), building an environment over a series of releases in FY 88 through FY 90. The environment would be built on top of the Common APSE Interface Set (CAIS) virtual operating system and utilize previous DoD MAPSE efforts.

The approach is tied to a family of development methodologies based on a conviction that software systems can best be built by following a disciplined approach that focuses attention on successive phases of the software life cycle. Most recently, this approach has been defined in several life cycle methodologies and formalized by the DoD-STD-SDS, a tri-Service software development standard currently being reviewed within the Department of Defense software community.

While approaches vary, each of these methodologies specifies that the details of a system's requirements should be developed initially, followed by a preliminary design and then a detailed design. These activities precede the implementation of the software and its integration into the system in which it is a component. The JSSEE approach is to develop a software engineering environment by following a life cycle-oriented methodology such as the methodology defined in DOD-STD-SDS.

A major issue is how common the resulting environment is with respect to the community as a whole. Even if the environment is viewed as universal, it will be necessary to augment it with organization-, project-, and application-oriented tools to increase its utility for any specific project. To the extent that the practices of any given organization are unique to that organization, then it may be necessary to view the common environment as just a basic set of tools to which organization-specific tools have been added. From a "core" it would be possible to create an environment common to all projects within an organization. Figure A-1 shows the commonality of tools based on an organizational hierarchy. The environment development project includes the basic tools and, in addition, some of the tools from the other layers. The other tools to be produced are common to some but not necessarily all of the organizations, projects and applications within the DoD.

APPLICATION-SPECIFIC TOOLS: technical and managerial tools specific to particular application areas

PROJECT-SPECIFIC TOOLS: technical and managerial tools specific to particular methods or management styles

ORGANIZATION-SPECIFIC TOOLS: technical and managerial tools specific to the practices and policies of specific organizations

BASIC TOOLS: technical and managerial tools that are of utility over all organizations, projects and application areas

**Figure A-1. An Environment Organization that Emphasizes Tool Commonality**

The approach being discussed here is, therefore, to first define the requirements for the environment that reflect a wide-degree of applicability over DoD organizations, projects and applications, in order to develop preliminary and detailed designs. The environment would then be implemented, tested and delivered to DoD organizations.

## 1.2 Objections and Responses

Objections to this approach have come from general concern over the use of a top-down approach for the development of large-scale software systems destined to be used in new and not fully predictable ways. Some stated objections to the JSSEE approach include the following:

-- the requirements cannot be fully defined at the beginning

-- too long a period of time will pass before a usable capability is achieved

-- it will be difficult to extend the system, in the future, to meet new requirements

-- the JSSEE will be too costly

-- it will not be easy to transport the environment to new host machines or new development situations

-- the capabilities will be either in conflict with the special needs of specific organizations, projects, or application areas or they will not be an adequate basis for augmentation to meet these needs

For the large part, these objections stem from a concern that fixing the requirements at the beginning will create an inflexible system that cannot be easily extended. And to a large degree, this concern, and therefore the objections that stem from it, can be addressed by careful planning that attends to the issues of flexibility and extensibility. This planning

and the technical issues that affect it are discussed in the following section.

### 1.2.1 Planning and Prototyping

The development of a flexible, extensible environment requires careful preparation for maturing the environment in the future. This, in turn, requires a system structure that can accommodate changing demands while still providing a high degree of coherency for the tools in the environment. This planning and the associated technical problems of prototyping and environment integration are discussed in this section.

### 1.2.2 Planning for Change

The central idea in planning for change is to identify the various changes that might occur and plan for handling them. We cannot hope to determine all of the potential changes. But a natural by-product of determining system requirements is a list of additional features that could be present but which have been rejected for the version under consideration. Such a list is a starting point for a more complete accounting of the changes that could occur in the future.

This accounting can be facilitated by developing prototypes of the system and letting the prototype users provide feedback on the system's requirements. This is an effective procedure for determining the set of compatible, necessary features to provide in the initial version. It can also be used to make a more complete accounting of potential future changes.

Accounting for potential change is less than half the problem. To actually respond to change requests in the future requires a system architecture that accommodates change and a solid basis for tools so they remain a coherent set even under change. These two technical problems are discussed in the next sections.

### 1.2.3 Prototyping

Prototyping is a way of determining whether a system's requirements are complete and consistent, but it can also be used to solve design problems such as developing a flexible architecture that can allow efficient change. Flexible architectures for software engineering environments have received little direct attention. In addition, it is unlikely that a flexible architecture for one situation will be sufficient without at least minor modifications for other situations. Prototyping for the purpose of investigating architectural issues provides the opportunity to empirically develop a flexible architecture tuned to the system's potential future changes.

### 1.2.4 Environment Integration

An environment is integrated to the extent that the tools provide a coherent collection. A very strong degree of integration can be achieved by having the environment support a specific development or in-service support methodology -- the methodology provides a degree of relatedness that makes all of the tools fit together coherently.

New tools or capabilities added to create subsequent versions will rather naturally integrate into the existing set of tools if they support the methodology on which the environment focuses. It is likely, however, that the integration will be lessened as the tool collection grows since new tools will tend to broaden the set of methodologies that the environment supports.

There are other bases for integration and these can be used in developing an environment so that future extensions will not destroy coherency. One other possible basis for integration is to define a small, coherent set of tool interfaces --new tools can then be integrated into the existing set by having them fit with respect to the defined tool interfaces. A

perhaps better basis is to have all of the tools utilize a common set of concepts -- for example, an Ada-oriented environment could have all of its tools be based on the idea of communicating sequential processes.

By picking one of these other bases for integration, particularly the set of interfaces, the issue of methodology support and coherency with respect to a methodology is not ignored. Rather, the approach chosen is the primary basis for integration and the coherency of the tools with respect to a methodology is attacked separately and secondarily. This creates a situation in which future expansion of the collection of tools will result in a new collection that is still integrated to some degree even though the new set may support an expanded set of methodologies.

## 1.3 Issues in Considering this Approach

The JSSEE team has prepared a plan that fits the overall STARS funding availability -- roughly $120 million for development of the SEE with some funding included from SEI to perform its role of integration, test, and release.

Even though it is feasible to use this approach to develop an advanced software engineering environment, this does not mean it is always desirable. Some of the issues and concerns that must be considered when balancing this approach with other alternatives are:

**Need for Immediate Capability.** This approach tends to lengthen the time until a usable capability can be achieved. Even though a side-effect of the use of prototyping could be the early emergence of a usable capability, it may still take longer to get a production-quality, usable capability.

**Tool Interoperability.** By driving the development from methodological or tool integration concerns, the result may not

necessarily easily accommodate tools that have been developed for other methodologies or for other approaches to integration.

**Level of Commonality.** While this approach can be used for a number of different levels of commonality, there may be better approaches especially if the level of commonality is low (that is, just the base level in Figure A-1) or high (that is, most of the higher levels in Figure A-1).

**Technical Feasibility.** With this approach, it is possible to specify an environment that cannot be built because of a lack of solutions to the technical problems that may arise. When heading towards a "traditional" environment that supports a "traditional" methodology for developing systems in well-known application areas, the probability of technical infeasibility is low. If there is a lack of experience or a high degree of uncertainty, however, this approach may not be a good one.

## 1.4 Summary

The JSSEE offers a long term approach to environments. However, a complete or "goal" JSSEE is not needed in order to have usable capabilities. A mid-term compromise might be the delivery of an augmented APSE such as ALS as a "core" JSSEE.

Although the JSSEE builds on existing technology, it is ultimately a high risk development. Its orientation toward Ada, planned incorporation of life cycle management capabilities and complex use of information interfaces increase the likelihood of development problems. The JSSEE is in essence a revolutionary approach. Futhermore, the JSSEE is a "from scratch" design; the current operational concept document being only the first step in setting its requirements.

The JSSEE is a Government-funded and Government-owned approach that offers a choice of implementors ranging from

competitively selected contractors to the SEI. Regardless of the implementor, the DoD will certainly contract to procure the JSSEE in much the same fashion as Ada and APSE's. Current projections are that at least two Services will GFE APSE/JSSEE to software developers.

The SEI and Services are being tasked with the maintenance and configuration management of the JSSEE. The current team-based design effort and the likelihood of transitioning the JSSEE to the SEI means that the whole effort from requirements definition to distribution will be centrally managed.

## 2. Service Laboratories

The role and capabilities of the Service Laboratories in the software development process have been studied intensively in recent years (see, e.g., (4))--most recently by the the Software Engineering Institute Study Panel (3). The Service Laboratory alternative is defined here as increased support to existing DoD Laboratories to provide funds to continue and speed the development of suitable software engineering environments.

### 2.1 Description of Approach

There are two modes in which existing DoD Laboratories could utilize increased support and resources to develop or aid in the development of advanced software engineering environments: the Laboratories could operate independently or they could cooperate along intra-Service or inter-Service boundaries. Intra-Service cooperation builds on the commonality of the existing procedures, regulations and acquisition practices within the Services. Inter-Service cooperation, on the other hand, while building on DoD-wide commonalities, such as the upcoming DOD-STD-SDS, might also be motivated by more varied concerns (e.g., the desire to cooperate on tri-Service initiatives or the common demands of such applications as

avionics, battle management or electronic warfare). Whatever the details of Laboratory organization and response, a number of factors need to be defined in order for the alternative to be workable. These factors include the following:

-- **Research:** Who will perform basic and applied research, carry out experiments, identify new technologies and conduct education and training?

-- **Development and Support:** Who will produce risk assessments, demonstrate the utility of new technologies, qualify tools and environments for military use, engage in advanced development and "productization" of environments, and oversee the maintenance and evolution of standard environments?

-- **Communication and Distribution:** Who will oversee the distribution of environments and services to users, maintain the computers and other physical resources and facilities, and manage the (possibly several) user communities.

While each Laboratory and Service is likely to adopt its own style in defining these basic operational parameters, it is useful to look briefly at a simplified but concrete example of how this option might be organized.

The Navy Material Command has adopted (5) the following model. In the Navy view the Laboratories operate in concert with top management of the Joint Logistics Commanders (this does not necessarily mean that individual Laboratories must cooperate and cannot operate independently -- it does mean that their operation should be coordinated). (See Figure A-2 for the partition of functions and responsibilities for the Navy Laboratories.)

| Function | Staffed or Managed by: | | |
|---|---|---|---|
| | Industry | University | Laboratory |
| Research | | X | |
| Dev/Supp | | | X |
| Comm/Distr | | | X |
| S/W Devel. | X | | |

**Figure A-2 Distribution of Laboratory Functions**

The major concepts in the model include the following. The developers of DoD software play a key role in this structure. By actual acquisition (e.g., GFE on contracts) or by remote access, the Laboratories utilize networks, mainframe computers and contracting mechanisms to distribute their environment to the MCCR software developers. The Laboratories also play a key role in the productization of new technologies. Not only do they qualify and demonstrate new technology, but they gather usage reports, problem reports and maintain both the environment and its documentation as well. The research function is assumed

almost entirely by external organizations such as universities (either through direct tasking or through funds directed to Service research offices).

It should be noted also that the Navy model of environment development is consistent with the establishment of a Software Engineering Institute (3). Many of the research and several of the development and support functions can be delegated to the SEI, (or its functional equivalent), while the Service Laboratory can keep the distribution and qualification functions that are closest to the developers' activities.

## 2.2 Objections and Responses

Several objections to the expansion of Laboratory roles have been raised. A creative approach to Laboratory management can overcome many of these objectives.

DoD Laboratories already have the charters, although not necessarily the funding, to carry out the work necessary to develop and implement software engineering environments. All MCCR computer programs operate in some type of environment, even if that environment consists of only an operating system. DoD Laboratories have been involved in the production and enhancement of some of these environments, for example the SHARE/7 operating system. A success that more closely resembles a software engineering environment as defined in this report is FASP (Facility for Automated Software Production), developed by the Naval Air Development Center (NADC) in the mid-to late-1970's (6). In general, however, the Laboratories have not provided programs with advanced, modern software engineering environments and a large scale effort by the Laboratories in this area does not appear to be forthcoming.

**2.2.2 Limited resources** Laboratory budgets are very constraining. The costs associated with developing and maintaining a new environment rise quickly beyond the costs of design and implementation. Maintenance, training and distribution costs can quickly dominate all other activities. Particularly for organizations that view their missions as support of Service R&D activities, this burden is too great to sustain itself for very long.

The issue of limited resources can be addressed by a cooperative effort among Laboratories. A partitioning of functions such as that envisaged by the Navy (see above) together with a pooling of capital resources (e.g., the construction of a broad-band computer network for Service-wide distribution of resources) can in fact leverage a comparatively small investment on the part of each Laboratory. That is, massive investments in duplicate resources can be avoided.

**2.2.3 Personnel limitations** The identification and implementation of high quality software engineering technology to integrate into a new environment requires highly skilled personnel. DoD Laboratories compete for this labor pool, but industry typically offers higher salaries. It is, therefore unlikely that an effective design and implementation team can be assembled in a DoD Laboratory without contractor support. Personnel limitations can be overcome by cooperative efforts between Service Laboratories and by using contractors.

**2.2.4 Resource duplication** Without extensive coordination between Laboratories, the duplication in resources engendered by this alternative would be prohibitively expensive. In hardware costs alone, the NADC efforts have accrued millions of dollars in capital equipment expenditures. Even if only half of the existing Laboratories required large mainframe computer support on the scale adopted by NADC, the cost in new equipment to the Government could be as high as $200 million. When the cost of

development is added in, resource duplication could push the incremental cost of designing and developing a set of new environments to well over $1 billion. Assuming that 70% of the life cycle costs of the environments are in maintenance and support, the total cost of this approach could be as high as $3 billion.

**2.2.5 Laboratory biases** The essence of this objection is that Service Laboratories, rather than being part of the solution, are part of the problem. It has been said that in their roles as chief scientific centers of their respective Services, they have advocated and perpetuated outmoded and ineffective approaches to software engineering. The goal of increasing the production of higher quality software cannot be achieved until this cycle is broken.

**2.2.6 The effort does not fit the Laboratory role**

Despite the charter of DoD Laboratories, some claim the basic work of designing and implementing a new software engineering environment does not fit into the role of a Laboratory that was invented to deal with physical and system-level engineering problems. However, the Laboratories have the advantage of flexibility that is difficult to obtain in commercial settings. As one example, consider the relative ease with which the top management of a DoD Laboratory can staff in response to a given task. If the functional equivalent of the SEI is also assumed then there is considerably more responsiveness in the Laboratory hierarchy than there typically is in the commercial sector (which is bound by the inertia of market demands).

## 2.3 Summary

The approach of allowing DoD Laboratories to carry out the bulk of the work in developing the environments that are needed for the next generation of military systems is probably not

feasible without significant contractor support. However, DoD Laboratories have played and will continue to play an essential role in environment development.

**Time:** DoD Laboratories can adapt to whatever time scales are appropriate. In responding to near-term and mid-term solutions, Laboratories may in fact be the favored option.

**Technical Risk:** Due to personnel and budgetary restrictions, Service Laboratories are most likely to adopt incremental, evolutionary approaches. Thus the Laboratory developments will tend to be of relatively lower risk.

**Design Methods:** Existing technologies--unless they have been tailored to the needs of the Laboratory are not likely to be adopted without considerable re-engineering. The approaches most commonly adopted by the Laboratories are therefore likely to be top-down.

**Funding:** The source of funds for the Laboratories is the Service budget. Within the Laboratory, support for such developments could be drawn from a mixture of IR&D and program funds.

**Implementors:** The technical staff of the Laboratories should carry out a great deal of the work although--in practice--support contractors and consultants will be tasked as needed.

**Ownership:** All rights to Laboratory-developed environments reside with the Government.

**Business Model:** This option involves no commercial activities beyond the contracting necessary to carry out predefined tasks and is therefore a Government business model.

**MCCR Acquisition Strategy:** MCCR projects using environments developed under the Laboratory alternative could acquire the environment as GFE or supplied from the Government on a "use at your own risk" basis.

**Maintainer:** The intention is that the support capabilities of the Laboratory will be applied to the environment. In some cases life cycle support contractors may be acquired.

**Management:** The relative autonomy of the individual Laboratories makes a distributed management of their efforts the only feasible approach. In some instances, the Laboratories may choose to cooperate and pool resources.

### 3. Off-the-Shelf Alternative

Some argue (see e.g., (7)) that in order to supply the DoD with the software development/engineering environments to satisfy planned requirements, those requirements must be levied on the design of the environment itself. This point of view has never been supported by engineering studies. It, in fact, ignores a significant alternative: certify for DoD use and support the continuing development of off-the-shelf (commercial and Government-sponsored) software engineering environments. Advocates of a commercial environment present four arguments: (1) some existing commercial environments are suitable or can be adapted for military use, (2) DoD should leverage some of the considerable investment that the commercial sector has made in software development environments, (3) commercial environments are, in any event, already being used for military software development, and (4) growing market interest rather than military needs will drive the development of commercial environments resulting in cost savings to the military if it adapts a commercial environment instead of developing a specialized military environment.

Similar to the purely commercial option is the option of Government-sponsored near-term Ada Programming Support Environments (APSE's). The adaptation of APSE's for DoD-wide use avoids many of the barriers to immediate adoption posed by the commercial off-the-shelf environments. For example, the

fact that an APSE is based on Ada eliminates the need to adapt to it. Futhermore, since Ada language compilers must be certified for DoD use, the issue of how the environment is to be qualified is not as significant. Additionally, there are activities underway to develop capabilities to evaluate and validate APSE's (see, e.g., (8)).

The choice of an APSE alternative gives rise to two further options: choose a Government-sponsored APSE such as the Army Ada Language System (ALS) or adapt a commercial APSE. The only difference between these two approaches lies in commercial practices such as who will retain rights to the environment and how the environment is to be distributed to users. As the Government retains more and more control of the environment, this option becomes more similar to the JSSEE and Service Laboratory options presented above.

## 3.1 Description of Approach

Key to the commercial alternative is the fact that the commercial sector has already designed, implemented, and placed into use software development environments of considerable power and sophistication. The risk of adopting one or more of these environments for DoD use is lowered since the technologies have gained some degree of maturity.

Before defining a specific approach to this alternative, the possibilities for adopting and adapting a commercial environment should be examined. Three possibilities for adopting a commercial environment are:

1. **License:** In this method, the Government acts as a broker for an environment or collection of environments. By a separate, and independent, Evaluation and Validation (E&V) (8) the DoD would approve a given environment for MCCR use and place it on a list of such environments. If a contractor

wished to use a commercially available environment, he would consult the list of approved environments and enter into a separate licensing agreement with the vendor of an environment. The Government is protected from the idiosynchracies of the development environment in three ways. First, the E&V increases the likelihood that the environment contains a standard "core" capability. Second, a thorough development Test and Evalution (T&E) program on the delivered MCCR software insures that the operational system will be supportable in any environment that supports the core capability (see, e.g., (9) for details). Third, vendors on the approved list have an economic incentive to improve their product.

2. **Purchase:** A purchasing agreement gives the Government rights to a software environment. The basic goals of this option are similar to the licensing arrangement described above, with the exception that the DoD becomes the distributor of a Government-owned environment. The viability of this option depends on developers being willing to give up a portion of their rights in the environment to the Government. It may be economically feasible to acquire more than one environment in this fashion. Alternatively, it may be desirable to let the candidate environments compete with each other, the winner to be chosen in a competitive "fly-off".

3. **Procurement:** This method allows the Government to be a co-developer of an existing system by a two-step process. First, a candidate or set of candidate environments is identified and a set of issues to be resolved by adaptation is constructed for each environment. Second, a development contract is let to

adapt each candidate environment for MCCR use. Rights in the resulting environment are then determined by the contracting instrument and need not be uniform across the candidates. Integrated systems consisting of both hardware and software could be candidates since the Warner Amendment exempts MCCR applications from the normal acquisition process for computers as specified by the Brooks Bill (Public Law 89-306, 30 October 1965).

Central to the success of an off-the-shelf commercial alternative (see strategies 1-3 above) is the adaptation of the environment. If an environment is not immediately suitable for MCCR use, an adaptation procedure must be initiated to "harden" the software environment. In most cases, this will turn out to be an extension of the commercial environment along one or more axes. Some of the possible extensions and the feasbility of making them follow.

3.1.1 Basic Extensions. These are fundamental changes to the core of the environment. They may require changes to the environment's kernel or to the operating system on which the environment runs.

3.1.1.1 Rehosting. This adaptation is very easy as long as the environment has been well-designed and has an inner kernel containing the host machine dependent parts of the environment. If the environment requires a special mechanism then it can be handled by building a virtual machine on the new host machine.

3.1.1.2 Data Management Systems. If the environment cannot handle the large databases expected in MCCR software development and maintenance situations, then a database manager may have to be introduced. Experience has shown that as software engineering environments expand, database management systems (or their functional equivalent) become necessary to

provide module to module interface, to spool information to project files, and to provide "quick codes" for a project. There are several possible approaches to installing an adequate database capability where none existed previously. These range from installing a commercially available database management system to acquiring a database machine. If performance issues dominate, a separate development may have to be undertaken to resolve outstanding issues, but the technology to be brought into play in this option is fairly well developed.

**3.1.1.3 Human Interfaces.** The engineering of the interfaces will depend on the exact environment under discussion. Some environments rely heavily on graphic interfaces, menus and pointing devices, while others are command language driven.

**3.1.1.4 Kernel Software Engineering Environments.** The Kernel Ada Programming Support Environment (KAPSE) and its proposed interface standard (CAIS) provide a basis for constructing a kernel environment. It is possible (a prototype may be constructed to confirm this) to use CAIS as a definition of a virtual machine and operating system. The job of moving an existing environment to such a machine is then reduced to a rehosting problem.

**3.1.2 DoD-Approved Languages.** Most state-of-the-art environments localize the dependency on programming languages being used into a subset of tools. A major exception to this rule of thumb is the Interlisp environment, although Interlisp has reduced the problem to one of internal representation.

**3.1.2.1 Ada.** Once compilers and programming support tools become available for Ada, they should be transportable to commercial environments. The internal language Diana also needs to be adaptable in this fashion. A number of environments (see the environments discussed in 3.3 below) can handle these changes easily. For more highly integrated environments, such

adaptation will be more difficult and would probably need some degree of research and development.

3.1.2.2 Other High-Order Languages. Languages such as Fortran, Jovial and CMS-2 have some of the same characteristics as Ada, but there are notable exceptions that increase the cost and risk associated with adaptation of environments. The first is the difficulty and cost associated with transporting the language. In some cases (e.g., Fortran) such transportation is supported by an extensive history and many tools to aid the engineer. In other cases (e.g., CMS-2), massive machine dependencies, a lack of broadly based user groups and relatively few tools make transportation much riskier.

3.1.2.3 Special Target Machines. The situation for specialized target hardware gives rise to many of the same problems discussed in 3.1.2.2. In addition, special downloading capabilities have to be provided.

3.1.3 Methodology Support. As alluded to in previous studies of the military software environment problem, support for DoD development and life cycle methodologies is an important factor in defining and approving an environment.

3.1.3.1 MIL-STD-SDS. The SDS provides a complete methodology and reporting system. Implementing an SDS-oriented software engineering environment would have the advantage of enabling subsets to be drawn for use in other (non-MCCR or civilian) applications. The principal requirements levied by standards such as SDS surface in document management problems. There is little doubt that any of the environments that are likely candidates for adoption can accommodate SDS.

3.1.3.2 Specific Methodologies. Problems will arise only when there is a basic incompatibility between the methodology being imposed on the development and the support

provided by the environment. As a rule, the most promising commercial environments depend little on methodological considerations. On the other hand, the more integrated the environment, the more methodology-dependent it appears.

**3.1.4 Tools.** The set of tasks supported by the tool set in the environment is an important consideration in applying the environment to an MCCR problem. The tools must be capable of supporting contractual requirements, for example.

**3.1.4.1 Absorbing Existing Tools.** This is a programming technology problem. If the environment supports the language in which the tool is programmed, then the tool can be adapted. There are a variety of sources for MCCR-related tools that could be absorbed.

**3.1.4.2 Absorbing Future Tools.** As with existing tools, the technical issues resulting from tool absorption are standard ones. The main difference between existing and potential tools is that engineering issues for future tool absorption can be resolved at very early design stages of the tool, reducing cost and risk.

## 3.2 Objections and Responses

Commercial alternatives are frequently dismissed as being unworkable in a number of dimensions:

1. Commercial environments are not suitable for MCCR applications, because the requirements are more stringent than those of other applications.
2. Proprietary barriers present long-term support risks of sole source reliance.
3. Unstandardized environments present risks of increased costs due to incompatibilities and proliferation.

4. The most attractive commercial environments will not be made available for MCCR applications because contractors are unwilling to relinquish even a portion of their rights to the environment to the Government.

However, software development environments, software engineering practice, computer software technology and the nature of the computer software industry have changed considerably since these objections were first formulated. It can now be documented that:

1. Not only are state-of-the-art commercial environments capable of supporting MCCR applications, there are commercial environments that are currently being applied in applications that share the extreme requirements of MCCR appplications.

2. The perceived proprietary barriers are not insurmountable. A creative acquisition strategy supported by a system of incentives can reduce or eliminate the risk of being bound to one supplier.

3. Proliferation is not a significant source of risk in software engineering environments. While standardization on source languages (e.g., Ada) and certain other aspects of the development process (e.g., MIL-STD-SDS) are essential risk reducing factors, the environment itself is not. In fact, deliverable specifications, and creative and well-managed software test and evaluation have been demonstrated to be the determining factor in the successful transition between development and support environments.

4. Not only are the best commercial environments available for MCCR applications, but many of the developers are currently willing to contract with DoD

for insertion of these environments into standard practice.

The commercial alternatives do not preclude a long-term plan for an advanced software factory. They do however present some attractive advantages: availability, low cost, and an opportunity to immediately use advanced technology. In addition, this alternative has the advantage of being consistent with the appearance of the SEI and the recommendations of the IDA-STARS Rights in Data Technical Working Group (10). It also utilizes proven technologies and proven technologies are essential for MCCR applications.

## 3.3 Discussion

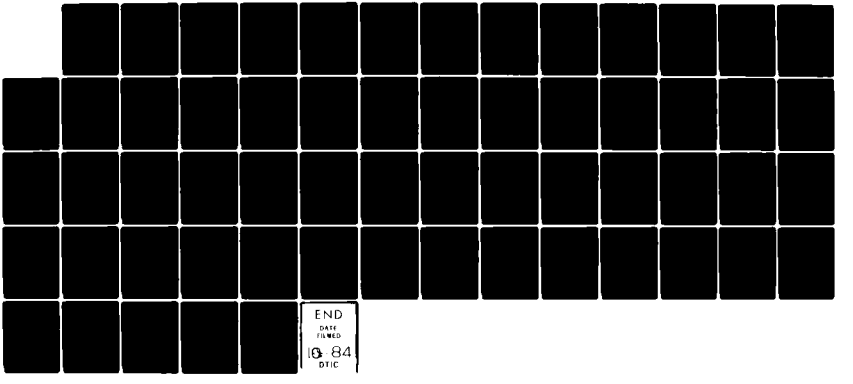
A critical objection raised in the previous section is the demonstration of the technology. In this section, we provide a more detailed response by surveying environments that may be candidates for inclusion in the off-the-shelf commercial alternative. These commercial-application environments, designed and developed for non-MCCR applications, require adaptation.

**3.3.1. Existing Environments.** Software environment research has grown and in some senses paced the development of software technology. This section surveys the capabilities of three development environments. These environments have been chosen to illustrate the range and variety of functions and architectures that are available.

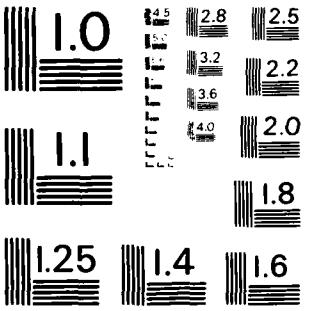
1. The Xerox Development Environment (XDE) is a highly integrated software development system consisting of hardware, programming languages, user interfaces, computer network and system resources.
2. Unix is a highly successful operating system that is gaining de facto standard status among 16-32 bit com-

AD-A145 285 SOFTWARE ENGINEERING ENVIRONMENTS FOR MISSION CRITICAL APPLICATIONS -- ST..(U) INSTITUTE FOR DEFENSE ANALYSES ALEXANDRIA VA R A DEMILLO ET AL. AUG 84 IDA-P-1789 2/2

UNCLASSIFIED IDA/HQ-84-28866 MDA903-84-C-0031 F/G 9/2 NL



END
DATE FILMED
10-84
DTIC



interfacing that encourages certain desirable software design methodologies.

3. The ITT Programming Support Environment (PSE) presented here is actually an amalgam of four development environments that has been constructed by ITT to construct and maintain the ITT 1240 Digital Switch. Although not one of these environments exhibits all of the tools and features discussed here, they do in concert present these capabilities. Key features of the PSE are its low cost and the degree to which the PSE uses off-the-shelf tools and integrating software.

These three environments have been chosen for examination for a number of reasons. The first is the relative maturity of the software. They have been used, at least in part, over a number of years and have acquired a sizable user community. Second, they lie outside the realm of standard MCCR environments. Third, these environments span the usable range of integration: from the highly integrated XDE to the loosely integrated PSE. The relative strengths and weaknesses of integrated versus non-integrated environments have been discussed elsewhere in this report.

One environment that we have used as an example other places but which will not be examined in detail is the Interlisp environment. This is mainly because the applications to which Interlisp is usually put. Interlisp is a laboratory software development environment that is soon to be available commercially. It supports complex research projects formulated in the Lisp language and has been used mainly for Artificial Intelligence applications in the past. It does, however, have the potential for supporting other languages besides Lisp.

### 3.3.1.1 The Xerox Development Environment (XDE)

XDE is the commercial version of Xerox's Mesa software development environment, a system of software engineering tools, aids, and resources that has been the standard software environment at Xerox since 1980. At a superficial level, XDE is based on the Mesa language which bears many similarities to the Ada language. Xerox software systems are destined for embedded computer applications involving many MCCR characteristics.

The most striking aspect of the XDE system is its level of integration. It is a total system built around a hardware base of workstations, bit-mapped displays and advanced graphics capabilities, Ethernet distribution of system resources, and shared system services. A disadvantage of this approach for MCCR applications is the inherent performance limitations of the workstations. Xerox plans to actively support the adaptation of XDE and interfacing of system capabilities to other classes of systems (e.g., VAX 11/780 class machines) and seems to be aware of the need for constructing plug-compatible interfaces with other environments.

XDE software consists of a collection of tools and interfaces that have been constructed into an environment. XDE tools are organized into the following categories:

1. **User Support Tools:** mail, file services and printing services.
2. **Mesa Language Development:** compilers, debuggers.
3. **Non Language-Specific Development Tools:** project management systems.
4. **Pilot Operating System:** virtual memory systems, process controller windowing.

Currently XDE consists of approximately 50 supported tools. In this environment "supported" is precisely defined and tied to

a release protocol--that is, a configuration management model that guarantees stability of supported services. A given environment may contain many more tools that are not supported or are in more fluid stages of development. In addition to the tools, XDE contains a great deal of "glue". The total size of the XDE is approximately 400,000 lines of Mesa source code.

In the ten years that followed the appearance of the Mesa programming language, the XDE evolved to address the following concerns:

1. construction of large embedded system software and production code for embedded computer products
2. support of evolutionary system development and responsiveness in situations where change is expected (incremental changes resulting in incremental costs)
3. integration of tools.

The resulting environment has been used for product development by two system development divisions at Xerox as well as by Versatek. By 1982, there were over 800 man-years of experience with the environment in the following areas:

1.5M lines of product code
0.7M lines of internal tools
1.0M lines of system prototypes.

Among the products constructed with XDE are the STAR office automation software, Xerox Network System, XDE itself, VLSI layout and design facilities, the Xerox 5700 printer software, a variety of CAD/CAM tools, and copier controllers.

XDE exhibits a number of design features that are often cited by users as unique and an inducement to heavy use. The design of the environment is based on a tool concept in which the tools play a passive, non-preemptive, and non-intrusive role. User interfaces are controlled by the user. In

appearance the user interface is based on windows and menus with a high-quality bit-mapped display and pointing device called a mouse as the chief communications media. It is possible to construct layers of tools and enforce cooperative protocols between layers, so that, for example, when more than one tool wants access to a file, access is granted in a reasonable way (e.g., during a recompile, active windows that are affected are also updated).

XDE offers support for other languages through its non-language dependent facilities and implementations of C, Fortran and Pascal are either in progress or are under discussion. Similarities between Mesa and Ada suggest that Ada implementations are not far behind.

XDE supports the system modelling and design tools needed for the construction of large systems (e.g., involving 6,000 files, 200 or more components, more than 50 programmers and at least 500,000 delivered source lines). XDE is also suited to projects in which development is distributed and parallel. The release structure is especially supportive in situations where the developer and the maintainer are not the same.

Essential information about the system being developed is contained in interface descriptions (i.e., connections between modules), compilation information, and file information as well as the mechanical support needed to manipulate this information.

One area in which additional development is needed is in analysis and reporting that would make little sense for commercial software but will be highly used in DoD projects.

XDE is currently workstation-based and runs on processors with large virtual address spaces. Users interact with hardware through its graphics terminal using keyboards and a mouse. Storage is by means of rigid and floppy disks. Users and network services are interconnected by a high-bandwidth

Ethernet. Xerox plans on releasing the source code for network protocols so that these networks can be interfaced to non-Xerox products.

#### 3.3.1.2 The Unix Programming Environment

The Unix operating system was developed at Bell Laboratories in the early 1970's to provide programming support to the developers of system software. Until the late 1970's it was widely distributed to the academic and commercial world on an unsupported basis. It has since become the system of choice for a wide variety of machines, most notably 16 and 32 bit workstations. Unix system III was offered in 1981 as a supported Western Electric product and a major system upgrade has recently been announced.

Unix can be viewed in two different ways. First, it is an operating system and as such it can host a variety of application systems ranging from word processing to embedded software development. Unix has had considerable influence on other operating systems in the last ten years. Many operating systems of this vintage in direct imitation of Unix or indirect imitation of Unix features have received general acclaim (e.g., pipes).

A second way to view Unix is as a software development support system. The vast majority of its more novel features are oriented toward this task and there are many success stories of large, complex software systems being built with Unix or one of its "look-alikes."

Unix reflects--but does not rigorously enforce--a specific philosophy of software development. A major initial goal was to allow software to be developed on small systems and then downloaded. Thus it fairly naturally supports teams working on the development of large-scale software systems with a relatively minimal investment in the system supporting the

development team. Whatever the scale of the system being developed using Unix, it is best to view it as a collection of small modules having simple interfaces since all of the system development tools in Unix are oriented toward this perception of software.

The Unix system consists of three major parts:

1. **The Kernel:** a basic operating system providing a file system and support for input/output, nested process activation and interprocess communication.

2. **Command Language:** a user control language that allows general programming concepts to be used in specifying the activation of processes.

3. **Tools:** programs that can be freely interfaced to perform some overall processing function.

Unix System III provides over 200 tools that carry out very simple tasks (like displaying the current time and date) as well as very complex tasks (like preparing a document for phototypesetting). Unix also contains tools that help in the building of other specialized or general purpose tools. An example of the latter is the tools called YACC (Yet Another Compiler Compiler) which is used for compiler construction. In addition to these vendor supplied tools, there are virtually thousands of tools available within the large Unix users community that has emerged over the last decade. Many of these tools are available at little or no cost.

The original version of Unix was implemented by one person in one year. Thereafter, it was re-programmed in a non-assembly language (so that it could be ported to a new machine) in an additional person-year. These preliminary versions contained the kernel, the command language (or shell) and a small number of tools. New tools were added as a result of its use during the next five years both inside and outside Bell Laboratories.

During this period it became widely used in the academic sector but its use in the commercial marketplace was limited.

In the late 1970's Unix was rehosted to VAX's and in the process a major redesign and upgrade was carried out. This was accomplished by one person in about 18 months and set the stage for an ever broader propagation of Unix throughout the community. Coincidentally, Unix caught the fancy of the commercial sector and its use began to spread into commercial and government arenas.

Very few goals have ever been explicitly stated for how Unix should support software development. The evolution of Unix has resulted in it exhibiting several attributes that are desirable for the development of large-scale systems:

1. the environment's host machine is not necessarily the same as the target machine on which the operational software executes,
2. a system is composed of modules, each performing one (generally simple) function and with a simple interface
3. new, perhaps specialized, tools can be easily generated and incorporated into the environment.

Unix itself is sufficient to support the development of software in many application areas. However, it fails to cover the life cycle and frequently needs to be enhanced to be of use in developing systems in specific application areas.

#### 3.3.1.3 The ITT Programming Support Environment (PSE)

The ITT Programming Support Environment is the collection of tools, methodologies and life cycle models used by ITT and its member corporations worldwide to design, construct and support the software for a family of digital switches known as the ITT 1240. In most of this discussion, the term PSE will be used loosely to apply to a collection of environments rather

than a single environment. In practice, however, a loosely integrated collection of capabilities such as the ones described here could be assembled from existing environments.

Unlike XDE and Unix, PSE does not have a coherent architecture to any useful degree. It consists of loosely coupled tool sets--the integrating mechanism is mainly to provide the host operating system and its utilities.

The ITT 1240 digital switch is the largest development project undertaken by ITT. The software effort alone is remarkable for its size, complexity and distributed nature. It is a multinational development with significant design and implementation stages scattered over several countries. The product supported by PSE is a digital exchange--that is, a variety of network modules providing telephonic functions. These modules are configured to respond to the requirements of a particular installation and may service phone lines for a building or a city of several million people.

The 1240 software is programmed in CHILL, a special-purpose language for telecommunications applications. The product life cycle supported by PSE consists of the following major stages:

-- product definition
-- design
-- implementation
-- analysis and testing
-- maintenance
-- termination

ITT has plans to transition the technology developed in support of the 1240 development effort into use on other products. Company spokesmen also expressed interest in

commercial ventures involving PSE, although resources for exploring a new software market appear to be somewhat limited at the moment.

The PSE environment is oriented toward the engineering of software components that can be individually tested and validated. These components are then assembled in off-the-shelf fashion into a completed switch by a separate product team. Still other groups are responsible for maintenance. The PSE provides tool support for this process of development-production and manufacturing-maintenance. Tools can be classified by the activities they support:

-- project management
-- system design
-- programming design
-- unit code and testing
-- integration
-- system testing
-- production, manufacturing and distribution
-- documentation
-- maintenance
-- change control

The ITT Software Methodology consists of a program design phase in which a nucleus of product software is created, followed by a customer development engineering phase in which the nucleus evolves and finally a customer application engineering phase during which the installation requirements of a final site location are integrated into the product.

Tool packages were developed to support these activities. These may be viewed as subsystems of PSE. The major support subsystems are:

1. **SDSS (Software Development Support System):** Consists of tools used to create program load modules from

CHILL source code and is used by development programmers and customer design engineers.

2. **IDSS (Integrated Documentation Support System):** This system is used for the creation and management of documents containing both text and graphics. Its main application is in the production of user documentation.

3. **CHARTS (Change Handling and Routing System):** This subsystem manages the tracking and status of software problem reports and the changes they induce.

4. **TEX (Text Executive):** This text harness/driver provides host level simulation of target systems and environments.

5. **LTF (Laborabory Testing Facilities):** LTF is the system integration test control software.

6. **SPMS (Software Production Monitor System):** This is the heart of the configuration management system, providing version management and control facilities and a database of product descriptions.

7. **PRISM (Program Information and Status Management System):** This system keeps track of software components for schedule and planning purposes and also manages interrelationships between modules for system integration purposes.

8. **CMSS (Configuration Management Support System):** This is the primary parts list database.

9. **CAESS (Customer Application Engineering and Support System):** This system accepts customer data for a specific switch and provides engineers with production information for that switch.

10. **SMSS (Software Manufacturing Support System):** This system aids in the assembling of the software for a specific switch from parts and specifications.

Currently, PSE consists of about 50 tools and tool sets. These tools are in the process of an on-going development effort that will result in a slightly more integrated environment. PSE is currently hosted on IBM 370/VMS systems, but the newer versions are targeted for VAX (Unix and VMS) computers.

In the early 1970's ITT realized that the design of its embedded computer software (already in the 500K delivered sources line/500 man-year size range) would increase in size and complexity by at least an order of magnitude. Planning for the 1240 product made these projections concrete.

To respond to the demands of software development on this scale, ITT established (in 1981) a center in Harlow, England to compile and deliver the tools needed to develop and manufacture 1240 switches.

ITT's overriding design philosophy is to view the PSE as an economically feasible way of integrating and coupling support tools. Interfaces are for the most part provided by the host environment. Tools are (whenever economically possible) purchased off-the-shelf or re-used. ITT has even entered into agreements with potential competitors to procure proprietary tools.

To promote maximum re-usability in programming environments, ITT has settled on both a standardized software life cycle and a production mechanism for programming support tools. The essence of the life cycle (the details of which are company proprietary) is to meet the special needs of ITT's products:

-- **Large Development:** 1240 is a billion dollar development

-- **Small Development:** European developments are small by comparison

- **Instrumentation:** The development process should be auditable

-- **Multiple Application:** Products such as the 1240 are used for many different purposes

- **Maintenance:** The maintenance problems of ITT products are unique (massive patching, relatively stable requirements, an evolutionary orientation to flagship products)

-- **Technology Migration:** The life cycle support organizations are not the same as the developing organizations.

The center in Harlow maintains a matrix responsibility for tool development. For example, a product manager may be responsible for a tool or group of tools and for meeting user requirements. The activities and interfaces identified for the delivery of tool services include the following:

-- Identification of User Requirements
-- Process Definition
-- Architecture
-- Decision to Build or Buy
-- Tool Development
-- Integration of Collections of Tools
-- Qualification
-- Distribution and Installation Support
-- Service and Maintenance

In short, the ITT design philosophy is a bottom-up strategy. Tools are the essential feature of its environment, and integrating mechanisms are defined at the tool level. The integrating interfaces to the tool sets have been constructed

using commercially available components. For example, the VAX/VMS implementation is built on a standard DEC product ("ALL-IN-1") which does menu management and tool linking automatically. Therefore the user can interact with ALL-IN-1 to determine menus and forms that will be required, standard VMS database protocols to access the file system and a communications interface. Even so, ITT has a considerable investment in PSE. The development of the tool set spanned five years (1977-1982) and consumed 500 man-years. An additional 100 man-years per year is required to support the tools. Even when the customer application engineering tools are not considered, the ITT estimates of the development cost of the PSE lie in the $10M-$50M range.

The exact contents of the PSE tool set is company proprietary, although the subsystem descriptions (given above) reveal the functions of the component tools:

1. **Management:** tools for costing, tracking, personnel management and tasking
2. **Development:** tools for design, interface utilization, development utilities, and a test environment
3. **Customer Development Engineering:** tools for evolving the product nucleus
4. **Customer Application Engineering:** the CAESS subsystem described earlier comprises these tools
5. **Maintenance:** these are tools for patching errors and modifications into the delivered programs.

#### 3.3.1.4 Ada Programming Support Environments (APSE)

Commercial and Government-sponsored APSEs are currently under development. The Army and Air Force have ongoing efforts to procure tools to support software development (principally coding and unit test activities) using Ada. Those tool sets,

known respectively as the Ada Language System (ALS), and Ada Integrated Environment (AIE), are in different stages of development.

The Ada Language System (ALS), whose first production release is due in January 1985, is an Army-sponsored effort to design, develop, test, and document a MAPSE (Minimal Ada Programming Support Environment) intially hosted on a VAX 11/780 computer. The ALS incorporates the following (12):

- Ada Compiler Machine Independent Section
- ALS VAX 11/780 Code Generator
- ALS VAX 11/780 Assembler
- ALS VAX 11/780 Linker
- ALS VAX 11/780 Listing Tool
- ALS VAX 11/780 Exporter
- ALS VAX 11/VMS Symbolic Debugger
- ALS VAX 11/780 Runtime Support Library
- Program Library Manager Tool
- Maintenance Aids Tools
- Container Data Manager/Program Library Manager
- Command Language Processor
- Environment Data Manager
- Session and File Control Tools
- File Administrator
- HELP Facility
- KAPSE

The Ada Integrated Environment (AIE) is an Air Force-sponsored effort to design, develop, test, and document a MAPSE including a state-of-the-art rehostable/retargetable Ada compiler. The AIE, which will be developed in accordance with general requirements specified in STONEMAN (February 1980), will be hosted on an IBM 4341 Computer System. The AIE incorporates the following components:

- Ada Compiler targeted to IBM 4341
- AIE IBM 4341 Program Builder/Linker
- AIE IBM 4341 Unit Lister
- AIE IBM 4341 Program Library Manager
- AIE IBM 4341 Change Analyzer
- AIE IBM 4341 Recompilation Minimizer
- AIE IBM 4341 Link Map/Cross Reference Lister
- AIE IBM 4341 Source Reconstructor
- AIE IBM 4341 Run Time System

- AIE IBM 4341 Symbolic Debugger
- KAPSE/Virtual Operating System
- Database Manager
- History and Backup/Recovery Facilities
- Configuration Management Package
- High Level I/O Package
- Mail Facility
- Command Language Processor
- HELP Facility
- Editor
- MAPSE Generation and Support Facilities (12).

For more than a year the environment and tools have been on hold while efforts have been concentrated on the Ada compiler that will run initially under the Unix-like UTS operation system.

Commercial efforts to develop APSEs include those by Telesoft and Data General. These APSEs are being developed to operate in concert with each company's respective Ada compiler.

## 3.4 Summary

The off-the-shelf options present a number of attractive features, including the opportunity for an early environment capability.

**Time:** This option provides near- and possibly mid-term solutions. The least integrated commercial environments -- exclusive of APSEs can probably be adapted within a calendar year. The most integrated would certainly require several person-years to adapt. Of course, the commercial options also provide the possibility of evolving over longer periods of time.

**Technical Risk:** Since the adaptation of existing environments is an incremental process -- i.e., capabilities can be added to the environment slowly -- the technical risk can be controlled. The only other significant source of risk is the possibility that the adapted environment may not be suitable for some MCCR applications. This risk can be minimized if the commercial option is combined with other strategies (e.g., application-oriented options).

**Design Methods:** Since the off-the-shelf options start with

an existing technology base and engineer MCCR environments from that base, the design method is bottom-up.

**Funding Source:** The source of funds can be either private or governmental depending on how the adaptation takes place. If the Government funds all or a portion of the adaptation efforts, then the Government investment probably leverages a 10 to 100 fold investment in environment R&D.

**Implementors:** Whether the environment was sponsored by the Government or commercially, the actual developer and implementors are full-time environment developers and vendors.

**Ownership:** The ownership of commercially-sponsored environments resides in the private sector. In light of current acquisition regulations, this may result in some loss of rights in the environment by the developer. Therefore, this option should be considered in concert with the Revising Policy alternative.

**Business Model:** This option assumes a standard model of business practices in which the technology is licensed to users under standard contracts.

**Acquisition Strategy:** The Government will specify the range of acceptable environments to contractors. The contractors' responsibility is to present a software environment that meets contract requirements. There are two possibilities. First, the MCCR developer can choose an already approved environment and arrange (e.g. by licensing) for the use of that environment with its owner. Second, the developer can offer his own environment for E&V (at the developer's expense).

**Maintainer:** The maintenance and support of the off-the-shelf environments should be carried out by their owners.

**Management:** The management of the off-the-shelf alternatives is decentralized. Once the necessary management practices are in place (which may be by centralized decision

making) - e.g., E&V procedures and streamlined procurement regulations - all other decision making is localized.

## 4. Sponsoring Commercial Development

This alternative is based on the approach adopted by the Very High Speed Integrated Circuit (VHSIC) program for increasing industrial readiness in hardware technologies. The key idea behind this alternative is to invest (probably competitively) in a small number of industrial organizations with the express purpose of increasing their software engineering environment capabilities. By combining this technique with either modified procurement practices (see, eg., Section 6) or one of the commercial alternatives, this Government investment would be transitioned to more widespread use.

### 4.1 Description of Approach

The degree to which existing DoD contractors can be expected to respond to this sort of option depends to a large extent on their readiness to provide environments that are suitable for investment and transitioning and their willingness to adapt proprietary environments for more widespread use. A number of factors enter into assessing the overall preparedness of industry to participate in this way.

**Methodologies**: A primary consideration is whether or not a given company uses a software development approach or methodology that is suitable for automation in an environment. Such methodologies may be characterized by the life cycle phases they encompass, the languages and applications that they are capable of supporting and the extent to which the methodology has been defined and formalized in standards, practices and procedures that can be monitored and evaluated. In many cases, although a company has in place such methodologies, these are

used solely for the software development of the organization, Government software development being covered by different methodologies to respond to standards and regulations that are included in contracts.

An important consideration in judging the maturity of a methodology is the maturity of its user community. As a first measure of this maturity, the operating history of the methodology is important. The number of successful uses of the methodology, the size of its user community and the complexity of software constructed under the methodology are all important considerations in making these determinations. Another concern is the extent to which the methodology has been allowed to migrate outside the originating organization. Corporations may find market advantages in widely publicizing their methodologies (either in standard scientific and technical outlets such as refereed journals or conference proceedings or in trade publications). More often than not, however, these sources are not suitable for transitioning a methodology to more widespread use. In cases, for example, where a "methodology" is really just a formalization of the procedures that have arisen in the parent organization, the best that can be hoped for is a "..here's how we did it..." exposition suitable for publication in internal magazines and newsletters but of limited external interest.

**Automation**: A company that seeks to expand its methodology using Government money may wish to enhance the extent to which the methodology is automated. The resulting environment or tool set is then a basis for future sponsored expansion. The questions surrounding the maturity of the automated methodology, its history on relevant projects, and the company's willingness to adapt and export the automated capabilities are important.

**Investment and Support**: A driving force behind the success of this approach will be the size of the investment needed to

draw a set of promising methodologies and environments to the stage needed for current and planned MCCR requirements. The fledgling environments may be developed under IR&D arrangements that are not suitable for Government investments. By the same token, a company may have commercial plans for developed environments and the acquisition of any Government-funded capabilities may deny them data rights that are crucial to business plans. Therefore, the dollar size of the Government investment may be less important than the overall climate in which the investment is carried out.

A basic strategy for implementing this alternative is based on an incremental investment, acquisition, and distribution of relevant environments. The critical stages of the process are discussed briefly below.

**Assessment:** This stage assesses the readiness of industry to provide environments. A competitive offering should be used to compile a list of candidate environments and methodologies for investment by the Government. Although it is expected that there will be a mix of mature and immature environments in which DoD will be interested, the key factor in carrying out this assessment will be the extent to which a company has already invested in its approach. This option, for example, is not intended to be a Government subsidy to the design and implementation of a "from-scratch" environment.

**Selection:** A small number of candidate environments are selected from the results of the competitive assessment. Factors to consider in this selection are the cost of the development effort yet to be undertaken, the extent to which the capability represented by the company advances the state of practice for DoD software development, the degree of technical risk involved in completing the environment, the willingness of the company to allow its technology to be transitioned, and the global assessment of how MCCR software development will be

affected by the availability or non availability of the environment over near-, mid-, and long-term horizons.

**Contract Award and Development**: Contracts are awarded for the completion and delivery of environments or tool sets that support the methodologies that have been selected. These development efforts should be evaluated and monitored using the experience gained by the VHSIC program.

**Transitioning**: The first uses of the developed environments will be critical factors in determining the success of the sponsored efforts. The best strategies may be to encourage competitive teaming arrangements on subsequent contracts. For example, if Companies A and B have received special development contracts for their environments, they may develop technologies that intersect each other to a considerable degree. The transitioning of each technology thus depends on the successful use of this technology on a DoD development effort. Company A may demonstrate its technology by bidding on a contract. Suppose however that Company B cannot bid on the same contract that A bids on. The contracting agent in this case may choose to "broker" a teaming arrangement between B and a third party that has the capability to respond competively with Company A.

A variation on this approach could be for the Government to acquire an environment through the standard procurement process. The Government could specify productivity and quality rates for the environment that are well in excess of current rates. There should be adequate lead time for the investment needed to develop practical approaches. The procurement could have separate specification phases and development phases and, perhaps, multiple participants in the specification phase. An RFI (Request for Information) asking for definition of what is required to achieve significant improvements could precede the actual procurement of the environment.

## 4.2 Objections and Responses

There are three primary objections that can be raised to this alternative. The first is that the approach is based on a false comparison between software and integrated circuit technologies. The second is that it throws Government weight and support behind a small number of contractors and will eventually result in the favoring of these contractors in improper ways. The third is that industry has not prepared itself and is seeking a Government subsidy for R&D programs that are best supported privately.

The essence of the first objection is that while the VHSIC program was designed to invest in and upgrade factories and facilities, the corresponding model for software can only result in the upgrading of capabilities. The key difference seems to be the degree to which software investments can be viewed as investments in capital improvements or in measurable industrial improvements. In the case of integrated circuits, the improvements can be quantified by the expense of fabrication facilities, new processes, and the increased functionality per unit cost of circuits destined for DoD applications.

Software presents a number of problems in this regard. First, measurable improvements are difficult to obtain. The most that can be hoped for with this alternative is the production of a software engineering environment. The productivity improvements and return on investment for any such environment are probably not predictable to any useful degree. Second, easily identified technological innovations would have to be essential outcomes of the selection and contracting process. Such innovations are rare and usually not identified until much later than their first appearance in a methodology or tool.

The second objection is difficult to counter. Any such strategy increases the capabilities of the company that receives the support in an unfair way. Government investment in Company A as opposed to Company B ultimately reduces to an endorsement of A's technology as opposed to B's. Furthermore, the fact that the Government has invested in A's technology makes it less likely that B will ever be able to "catch up." The danger is that such a situation forever locks B out of the Government contracting cycle for software intensive systems since A will always be one version ahead of B in its enviornment. That is, even though Company A may deliver a completed environment to the Government as a result of its contract, it will retain and upgrade for proprietary purposes those aspects of the environment that give it a commercial advantage on subsequent procurements, even though such capabilities were developed mainly at Government expense. This has the ultimate effect of limiting competition and channeling technology into a few relatively narrow paths. In a climate in which the basic technology issues are yet to be determined, this seems to be an unwise policy.

The third objection questions the apparent state of industrial preparedness to respond to such an option. These are detailed in the following section.

## 4.3 Status of Industrial Environments

To provide some indication of how prepared industrial organizations are to participate in this sort of program, seven organizations were contacted and queried about the status of their software engineering environments. Four of the seven were large corporations based in the same industrial sector. One organization was a company in a different industrial sector, and two were vendors of software ahd hardware products and services. Four of the seven were primarily Defense contractors.

The groups were questioned about three aspects of their environment capabilities: (1) methodology, (2) automated environments, and (3) the extent of external (Government) investment that would be desirable.

Six of the interviewees claimed to use some sort of methodology for software development, although half of these could only cite the Yourdon methodology as one that is consistently adopted. Others either had plans to develop a methodology or confused a methodology with the life cycle model in effect in their organization. Those groups that were able to cite documents that define the methodology (3) also protected those documents as proprietary.

Experience with methodologies varied widely. One user of the Yourdan methodology had applied it on a large DoD contract involving more than 80 programmers and claims a user community of hundreds of programmers. Others were not sure that their methodologies had ever been rigorously applied. No group reported documents or experience with a life cycle methodology. Six of the seven had plans to "market their methodology".

Only one of the seven claimed any degree of automated support for their methodology; that one turned out to be a user interface capability. All other sources built environment capabilities and tool sets by a combination of in-house tool development and off-the-shelf purchases. None of the organizations queried had a history of or plans for creating an environment architecture. In general the capabilities provided were more loosely coupled than the ITT PSE environment and did not even attempt to support life cycle models or development methodologies.

All of the groups expressed an interest in Government support for their tool development effort. It was clear, however, that tool investments would not be used to <u>design</u> an

environment but rather to enhance a pre-existing tool capability. No cost estimates were provided by the interviewees, but all noted that in the absence of direct DoD support, their efforts would be accommodated under IR&D funding. Surprisingly, even those groups that claimed they would take Government support were prone to claiming the rights to products of the R&D effort.

### 4.4 Summary

It is clear that an investment could bring about a rapid growth in the tool capabilities of selected contractors. Futhermore, this option is low risk since the industrial groups apparently rely on demonstrated technology to meet their needs. On the other hand, the approach is bottom-up to the exclusion of innovative technology.

Obviously, Government dollars are sought to fund these efforts, however the developers and vendors of the tools that are most likely to be included want to claim that the Government investment is in the "research" not in a product; in particular the companies wish to retain all rights in their products, making them available to the Government only under the protection of a contract to procure operational software.

Since the Government exercises almost no control over the internal choices of tools, there is a high degree of decentralization in the management of this approach.

## 5. Application-Oriented Environments

To the extent that environments built under the previous alternative (Sponsoring Commercial Development) cover a variety of applications, they will be horizontally integrated over the application area. Another alternative would be to select a single application area and build an environment specific to that area.

## 5.1 Description of Approach

There are several ways in which this alternative could be addressed. In the world of DoD software, it seems most sensible to capitalize on the capability already existing in large Defense contractor organizations by having them prepare environments tuned to an application area for which they have proven expertise and experience.

The approach would be to have several Defense contractors each build environments that are specific to particular application areas. These would then be consolidated into a single environment covering a variety of application areas.

## 5.2. Objections and Responses

The major objection to this approach is that there would be a definite lack of commonality among the environments, making it difficult or even impossible to integrate them into a single environment or a small set of alternative environments. Other objections are:

-- the environments might include proprietary information that would be unavailable to other elements of the DoD community

-- the environments would not attend to the problems of acquiring software but only to the problems of developing and supporting software

-- there would be a fairly high degree of duplication of effort that may not be justified in terms of the risk involved.

-- software engineering environments that are not methodology and life cycle driven may fail to be general enough to gain widespread use, resulting in a proliferation of environments.

The problems of ownership and availability of the produced environments have to be attacked by innovative approaches to Government/industry rights in data issues. That these problems

can be solved in a way that is acceptable to both the Government and to industry has been argued elsewhere (10).

The other objections can be addressed as they were in the previous alternative, by developing a reasonable plan (such as was developed to sponsor commercial development of the VHSIC program) that recognizes and tries to minimize the problems that may occur. The next section outlines such a plan.

## 5.3. A Scenario for Acquiring Application-Oriented Environments

The major problems that must be addressed are the duplication of effort and the inability to integrate the produced environments. One approach to solving these problems is to develop the environments in phases, gradually reducing the amount of parallel effort.

Several application-oriented environment development projects could be started in parallel. The goal of each would be to define, design, implement, demonstrate and put into actual use a complete, integrated environment that supports a specific project management technique, a specific full life cycle methodology for the development and maintenance, and the creation and evolution of software in a specific application area.

Figure A-3 depicts the appropriate organization of the environment to foster commonality. This would not necessarily be the physical organization for the environments; it is intended to be a logical organization that will force conscious decisions about the commonality of tools.

---

APPLICATION-SPECIFIC LAYER: tools specific to the chosen application area

---

GROUP LAYER: tools supporting the chosen methodology and the chosen management approach

---

CORE LAYER: generic tools, tools supporting tool integration and interoperability, and tools supporting environment extensibility

---

BASE ENVIRONMENT: a specific primitive environment chosen to foster portability and provide common facilities; could be one of the MAPSE's currently under construction through Government support (i.e., the ALS)

---

**Figure A-3: An Organization for the Multiple Development Efforts**

These projects would have the following four phases.

**Phase I:** define a specific environment oriented towards a specific application area, a specific development and maintenance methodology, and a specific project management technique

**Phase II:** design the environment

**Phase III:** implement and demonstrate the environment

**Phase IV:** refine the environment and put it into actual use.

During Phase I, each developer would prepare the environment's definition and, in addition, would specify how it aids the acquisition of software systems. The developer would also identify other application areas where the environment could be used, analyze the environment's completeness and degree of integration, sketch a design for the environment, prepare a brief plan for implementing the environment, and outline a brief plan for demonstrating the environment and putting it into actual service.

At the end of Phase I, a public review would lead to a potential reduction of effort. This review would identify potential commonality among parts and decide which of the parallel efforts should carry on the development of common parts. Phase II would result in a design for the environment and brief plans for implementing and demonstrating the environment and transitioning it into actual use. Another public review would be held to not only scrub and homogenize the designs but also to decide which of the parallel efforts would proceed further for the common parts.

Phases III and IV would proceed to develop and demonstrate the specific application-oriented environments utilizing the

commonly built parts.

This skeletal plan would have to be refined for actual execution, but it provides a basic approach to developing application-oriented environments that reduces duplication of effort and increases the probability of intergrating resulting environments.

As before, feasibility of this approach does not necessarily imply desirability. In comparing it to other approaches, there are several issues and concerns that particularly must be addressed.

- **Degree of Commonality**
- **Importance of Proprietary Rights In Data**
- **Feasibility of "Telescoping"**

Public reviews and the resulting reduction and coordination of parallel efforts may not be justified or feasible. The overall time period available for the development of the environments may not be adequate for review and may therefore not justify the approach. The reduction and coordination of parallel efforts may perturb normal staffing practices for the contracting organizations, rendering this approach infeasible.

## 5.4 Summary

If industry assessments of how close contractors are to providing application-specific environments are correct, then this approach provides near-term and mid-term options. The technology is an incremental improvement of what is currently available, and, since it builds on existing capabilities, is a bottom-up construction of an environment capability and a low risk approach. Other important parameters that distinguish this option are the following:

**Funding Sources:** The exact determination of funding sources depends on a number of other parameters. The range of possibilities is from leveraged Government investment (as in Section 4) to pure commercial development (as described in Section 3).

**Implementors and Maintainers:** In all cases, the software developers (contractors) are the implementors and maintainers of the environments.

**Ownership:** In this option, ownership, and therefore maintenance, of the completed environments resides with the private sector.

**Business Model:** The most natural business model for this option is an adaptation of standard commercial practices (either contract or commercial leasing/licensing depending on the source of funding), providing Evaluation and Validation support and insuring supportability of the delivered software by appropriate Test and Evaluation.

**MCCR Acquisition Strategy:** Since ownership of environments developed by this method will reside with the private sector, an appropriate acquisitions strategy would be to specify interfaces.

**Management:** The management of this alternative can be distributed among DoD Laboratories and other sources of scientific guidance on application-specific matters and therefore is decentralized.

## 6. Revising Policy

Unlike the other alternatives described in this report, the option of revising policy requires a purely administrative and policy oriented approach. The implicit assumption is that existing technical and market forces are sufficient to produce an advanced environment for DoD use, but that the current climate in the Defense acquisition community works against such

forces. The solution is therefore to modify the acquisition climate so that the natural forces can work.

## 6.1 Description of Approach

Policy for acquiring software for weapons systems in the DoD is defined in DoD Directives 5000.1, 5000.28, and 5000.29. (Although some MCCR applications formally fall outside the realm of these acquisitions, the policies are invoked in practice more often than not). These policies are supported by a vast array of directives and other policy vehicles as well as regulations and standards that serve to implement the policy.

Some key features of existing policy include the following:

1. The goal of DoD acquisition policy is to insure the efficient and effective acquisition of systems that are operationally effective.
2. In most cases, the management of the acquisition is to be decentralized.
3. Design and price competition insure cost effective development and responsiveness to mission needs.
4. "Readiness" and suitability of the acquired system are characteristics that are as important as schedule and performance objectives and operational effectiveness.
5. Stability in the acquisition process should be insured by planning and initiating low risk acquisitions and, realistic budget estimates.
6. Ownership costs must be balanced against acquisition costs and system effectiveness.

It has been the sentiment of several recent study groups that in the context of developing support software such as environments the acquisition regulations and particularly the rights in data clauses in those regulations are not consistent

with the policy goals stated above and should be modified in several innovative ways.

A "Rights in Data Technical Working Group" (RDTWG) (10) reported the following:

> ...the Government is failing to obtain the most innovative and creative computer software technology from its suppliers. Thus, the Government has been unable to take full advantage of the significant American lead in software technology for the upgrade of its mission critical computer resources...

The RDTWG further reported that the principal reasons for industry reluctance to offer its best products to the Government include the following:

1. Excessive data rights claims by the Government expose vendors to potential losses of proprietary rights in commercially valuable technology.

2. Government software acquisitions, contracts, and legal vehicles are unnecessarily complex and unclear due in part to a concern for maintenance and supportability that are not adequately addressed by current Government practices.

3. The active and profitable nature of the commercial software market biases it against the Government which seeks to maintain a set of business practices that is incompatible with normal commercial practices, tending to color the Government as a "bad customer."

The approach recommended by the RDTWG and other subsequent study groups (see, e.g., (11)) includes the following features:

1. Reduce objectional data rights claims by the Government by instituting a system of commercial licensing practices; at the same time, modify relevant acquisition regulations to insure that these incursions are not mandated.

2. Provide a system of regulations, standards and interface specifications that is sufficient to allow the specification of MCCR software support environment characteristics in RFP's. Require validation of software deliverables against the specified environments during test and evaluation.

3. Establish a Software Acquisition Board to oversee the process.

## 6.2 Objections and Responses

These options have only recently been proposed, and a coherent set of objections has not yet emerged to them. However, in their deliberation, the following objections to initiating any policy level changes have been encountered.

First, changes in the acquisition process for software would create an imbalance in the acquisition of hardware and software. The regulations and practices for the non-software components of the system would be oriented toward one set of objectives and carefully managed while the software would be governed by an inconsistent "free market" approach.

The hidden assumption in this objection is that somehow the manufacturing of the software is comparable to the design of the non-software components. In fact, the current climate is one in which imbalances abound (9, Vol. I). The goal is to improve the development and engineering of operational software. In this light the development environment is the "factory" in which the software is manufactured. The Government claims on factory technology that is not related to the operation and support of the operational software would be sharply curtailed in this approach.

Second, as a matter of policy, the Government cannot enter into commercial agreements such as licensing agreements. This

is, in fact, false. Through GSA practices, the Government routinely enters into commercial agreements that closely resemble standard business practices.

The Government needs rights in data to development software in order to insure the maintainability of the delivered software and to avoid being tied to a sole-source supplier of software and services for the life of the delivered system. This objection has been dealt with elsewhere in this volume (see, e.g., Section 3).

The final objection that has been raised against policy related approaches such as the one advocated here is that such approaches encourage gold-plating by contractors. The key to this objection is that the "best" development environments are not needed -- only environments that are sufficient to get the job done. If acquisition practices are modified too drastically, new elements of risk are introduced that affect the stability of the acquisition. The response to this objection is that evolutionary forces by themselves are not sufficient. The job that will be required of the software in the next generation of DoD systems cannot be built without access to the newest technology available in the private sector.

## 6.3 Standardization

Included in this option and others could be the development of standards that serve to coordinate the distributed, independent activities of the tool building community. The feasibility of this approach is argued in this section.

### 6.3.1 The Approach

The central theme of this approach is standardization of the methodologies and the interfaces among tools. By specifying the activities to be performed, the techniques to be used in performing them and the interfaces among the tools

that support these techniques, tool development itself can proceed in many ways at many sites and the resulting tools could still be consolidated to provide useful and effective environments. The first step would be to define standards for development, the methodology to be used on all MCCR contracts (life cycle oriented), and the interfaces among the tools that support the methodology. The second step would be to develop a mechanism for certifying that tools meet these standards and therefore could be made available for use in an environment. The third step would be to evolve the standards to reflect new software technology and upgrade the collection of tools that are certified.

### 6.3.2 Objections

There are several objections to this approach:

- -- the lack of more direct coordination leads to unnecessary duplication of effort
- -- few organizations will be able to produce certifiable tools
- -- the task of certification is too difficult
- -- either the standards will be too strict, leading to few tools achieving certified status, or too lenient, leading to a lack of integration in the collection of tools

As with the previous two options, these objections can be addressed by providing a plan that tends to minimize the negative effect of various problems. Such a plan is discussed below.

### 6.3.3 Maintaining a Pool of Tools and Tool Piece Parts

For any specific project, the software engineering environment will contain tools oriented toward the specific

application area for the system being built and the methodology and management practices used on the project. It will also contain generic tools that are relatively independent of the end-application area, the methodology, and the management practices.

One way to prepare an environment for a specific project would be to select applicable, existing tools, develop new tools to make the overall collection complete, re-engineer the tools (if needed) to integrate them, and install the resulting collection on some base environment. This process requires both a pool of candidate tools and a selection mechanism by which tools in the pool that are of value to the project can be identified. The approach would be considerably more effective in the long run if there were a way to feed the new tools back into the pool, after certifying them to some degree, so that they could be used in preparing other environments. This is diagrammatically presented in Figure A-4.

This process of building specific environments could be carried out in more than one phase. It might be of interest to a contractor, for example, to select a broader range of tools than would be used on any specific project and form a "local pool," specific to its organization and its practices and policies. Intermediate pools such as this could also be oriented towards specific application areas.

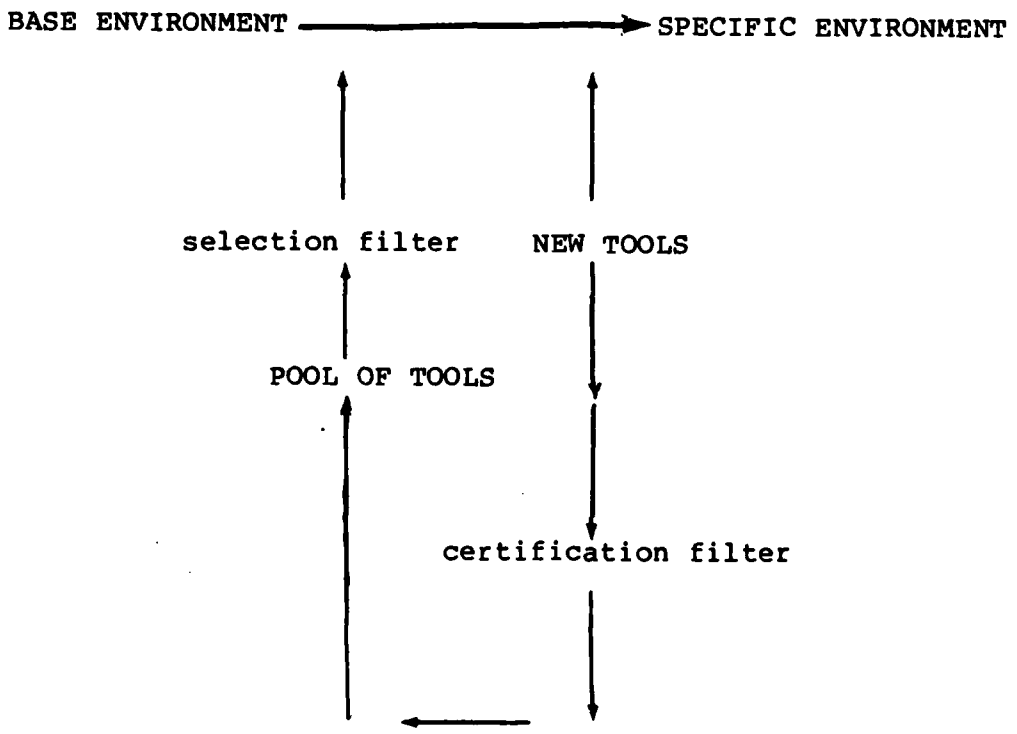



Figure A-4: Preparing Specific Environments Using a Pool of Tools

The individual elements of such an approach already exist. Any one of a number of available environments, from the relatively primitive Unix environment to the more extensive DCDS environment developed at TRW, Huntsville, could be used as the base environment. In addition, the Government-sponsored efforts to build a MAPSE (that is, the ALS, AIE, and ALS/N efforts), as well as several on-going commercial efforts, will result in environments that could be used as a base environment.

Something such as the NBS (National Bureau of Standards)-developed tool taxonomy could be used to provide a reasonable, albeit rough, organization for the pool of tools. The standards themselves will provide a first approximation to the certification filter. And an initial selection filter could be provided by the tool classification characteristics used in the NBS taxonomy or in the commercially available tool "catalogues" provided by several companies. Finally, much of the technology for fitting the pieces together to form a specific environment exists as a result of projects such as the Toolpack and Gandalf projects.

The success of this approach is somewhat dependent on breaking the large tools that we typically think in terms of today, such as compilers, into tool piece parts. If this is not done, then the wide variability of requirements for specific environments will result in few of the tools being selected and the need to build a large number of new tools.

For many traditional tools, the piece parts into which they should be broken are fairly obvious just from our general experiences. For example, breaking a compiler into a lexical analyzer, a parser, a code generator, and an optimizer is an obvious division that provides piece parts of general utility. This general knowledge that we have from working with traditional tools can be effectively used to break up a large variety of different types of tools into reusable piece parts.

While primitive technology for all the parts of such an approach is available, truly effective use of this approach will require several advancements. All of these seem, however, to be dependent on relatively straightforward developments rather than major conceptual breakthroughs.

### 6.3.4. Issues in Considering this Approach

Again, the desirability of this approach requires the consideration of several issues.

**Time Required for Standards Approval.** The time to have a set of standards approved and accepted has generally proven to be lengthy. Sometimes it is an easier and shorter process to provide a de facto standard. The need for explicit or tacit approval must be addressed when considering this approach.

**Need for Extensibility.** This approach provides a very high degree of extensibility at some extra cost, namely the cost to re-engineer and fit the various tools and tool piece parts together. The extra effort required to produce each specific environment must be weighed against the need for the increased degree of extensibility.

**"Proliferation".** The appearance of proliferation may be at odds with policy that dictates agreement on and support of commonality. The fact that such policy usually lies behind moves towards standardization is an anomaly of this approach. The need to educate policy makers so that this approach can be accepted as consistent may be too great an effort.

## 6.4 Licensing

The major barriers to effective modification of acquisition practices are the rights in data clauses in acquisition regulations. These regulations can be modified to permit contracting officers to enter into licensing agreements with

software vendors. A report proposing reform of Government rights in data clauses (11) sketches the mechanics of such a scheme.

1. Under such a license the contractor maintains most rights to software -- even if that software was developed in part at Government expense.
2. The license grants the Government the right to use the software solely for its own purposes.
3. For software developed solely at the contractor's expense, the developer would receive a negotiated royalty for use (no royalty would be possible for software developed at the Government's expense).
4. Rather than disclose proprietary information covered by a licensing agreement, the Government can direct the holder of the information and the inquiring party to enter into a direct licensing agreement under which the information can be offered with remuneration protection accorded to the holder of the information.
5. The terms of the license should include the steps which the contractor can take to protect any proprietary information and offer the possibility of seeking damages done under breeches of the license.

These practices can be carried out in the context of current acquisition policy, provided that the supporting regulations are modified and that standards are sufficiently strong to support this interpretation of policy.

## 6.5 Summary

Since this alternative, revising policy, is administrative, the time delay in implementing it is only as long as the review cycle for such changes. In addition -- since this option is compatible with many of the technical options given elsewhere in

this report -- the risk is as low as the technical options. Adoption of this set of recommendations since they are administrative, does not introduce any technical risk.

The cost to implement this alternative is limited to that of modifying policy, and the issues of who is the implementor, owner and maintainer of the environment are the same as for the commercial alternatives. The chief area in which this alternative differs from previous ones is in the development of new acquisition strategies.

## 7. Industrial Consortium

Another alternative approach to making significant improvement in the software state-of-practice for DoD mission-critical systems is to establish (or encourage the establishment of) a software consortium to build a software engineering environment. This section defines the concept of a software consortium and describes how it might operate. It explores the feasibility of establishing such a consortium and addresses the issues and the tradeoffs that should be considered.

### 7.1 Description of Approach

A consortium is defined as some type of arrangement or joint activity for effecting a special purpose venture requiring extensive resources to carry out. These extensive resources, whether financial, personnel or capital, are usually more extensive than any of the consortium participants desires to carry or expend alone. In other instances, each consortium participant brings a particular expertise to the venture and none of the participants alone possesses all the expertise required to carry out the mission.

There are various models for the organization and operation of consortia. Several examples of consortia currently exist in the engineering/computer science area; this section will

describe two of them: the Microelectronics and Computer Technology Corporation (MCC) in Austin, Texas, and the Semiconductor Research Corporation (SRC) in Research Triangle Park, North Carolina. These are just two of a wide range of possibilities for the operation of a consortium. In addition to the consortium model, numerous other arrangements can be set up, including R&D partnerships, joint ventures, and joint Government/industry/university teams.

Government/industry teaming represents a model that both the European community and the Japanese have taken for their high technology joint efforts. As an illustration, the ESPRIT (European Strategic Program for Research and Development in Information Technology) Project pools the research efforts of a dozen European electronics firms involved in the development of chip technology. The participants, including West Germany's Siemens, France's Honeywell-Bull, Italy's Olivetti and Philips of the Netherlands, will share in pre-competitive research that can be exploited for different marketing purposes once products are developed. The ESPRIT Project will be funded jointly by the European governments with an equal amount being shared by the companies.

### 7.1.1 Microelectronics and Computer Technology Corporation (MCC)

MCC was incorporated in August 1982 and announced in May 1983 that it would locate in Austin, Texas. MCC is a unique consortium in that it was set up as a for-profit corporation and received Justice Department approval as such. It was the first for-profit company to receive that approval. Eighteen companies are shareholders (member companies) in MCC: Advanced Micro Devices, Allied Corporation, BMC Industries, CDC, DEC, Kodak, Gould, Harris, Honeywell, Lockheed Missile and Space, Martin Marietta, Mostek, Motorola, National Semiconductor, NCR, RCA, Rockwell and Sperry.

MCC was set up to conduct advanced long-range research in microelectronics and computer sciences. Their research will be in four major programs:

- o Packaging
- o Computer-aided design (for VLSI)
- o Software technology
- o Advanced computer architecture
  - Parallel processing
  - Advanced data base architecture
  - Expert systems
  - Human interface

Specifically, the MCC software technology program is a six-year program whose overall goal is "to increase productivity of the software development process by one to two orders of magnitude." The research that will be pursued as part of that program includes knowledge based software development, natural language processing and improved validation and verification technology.

The mix of personnel at MCC is approximately 60% direct hires and 40% on temporary assignment from shareholder companies. These latter individuals can be assigned for periods of time ranging from one year to ten years. Most of the research programs are set up for five to seven years and are expected to produce research results at that time. Shareholder companies benefit in two major ways. They can fund one or more of MCC's research programs. When an innovation emerges, the companies that funded that research have exclusive access to that technology for three years. After three years, the technology is available to the entire industry through licensing agreements. The fees that result from the licenses are then split between MCC and the companies so that companies receive a return on their investment.

MCC has bi-lateral data exchange agreements with the Federal Government. However, MCC will not perform classified research nor will it compete for procurements. If a Government agency is interested in accelerating a particular research program at MCC, MCC would accept the Government agency as a shareholder "company". To date, this has not taken place.

### 7.1.2 Semiconductor Research Corporation (SRC)

SRC was the result of discussions involving major members of the semiconductor industry--Motorola, National Semiconductor, Intel and Advanced Micro Devices--and systems houses, including IBM, Digital, Honeywell and Control Data. In the early 1980s, semiconductor industry leaders realized that long-term research on a cooperative basis was becoming imperative. SRC was the result. It was incorporated on a not-for-profit basis in February 1982. It currently has 28 members in the semiconductor, systems, chemicals and equipment industries. SRC has both research and development programs underway. One of its targets for the 1990s is a commercial chip with one million gates, 0.5 nanosecond delay and 50% yield--a supercomputer on a single chip.

The SRC research is focused in three fields: microstructure sciences, design sciences and manufacturing sciences. Approximately 50% of its research funds are allocated to the microstructure sciences centered at Cornell University. In each of the three research areas, there are specific target goals to focus the research.

Although initially established to sponsor research in the universities, it has recently decided to establish a development program. In the past, development was a collaborative effort between semiconductor manufacturers and individual equipment makers. However, in the last decade, the complexity of equipment, development costs and selling prices have all

increased by an order of magnitude. Over the next five years, the cost of development and production is expected to increase by a factor of five to ten. In addition, fabrication at the submicron level is so complex that no single equipment manufacturer has sufficient expertise to "do it alone".

The SRC development program, as proposed, will be carried out in an in-house facility with both its own staff and people on assignment for one to three years from member companies. It is planned to be a four-year, $150 million effort with intermediate deliverables.

As an example, the cost of carrying out the cooperative R&D efforts at SRC is approximately $300 million: $60 million for research, $120 million for generic development and $120 million for automation of manufacturing processes. If U.S. semiconductor sales are about $12 billion per year, the proposed SRC budget for R&D amounts to 2.5% of sales.

There are three major differences between SRC and MCC. The SRC is a not-for-profit corporation, it has a more focused research program and uses grants to external organizations, primarily academic institutions, for the conduct of the research (although the development program set up recently will be performed in-house). The MCC is a for-profit corporation, it has a broad program of research in microelectronics and computer science, and performs its research in-house.

## 7.2 Objections and Responses

The specific target of this alternative would be the production (or at least the research underlying the production) of a software development environment that supports the entire mission-critical system life cycle. The participants would be primarily from industry (computer manufacturers, aerospace/defense contractors, systems houses, software houses) as well as from academia. Such a cooperative venture in pursuit

of a challenging goal could bring together the variety of expertise (that currently exists distributed in industry and academia) needed to build a software development environment. Interaction would be on-going with the Software Engineering Institute in order to carry out the technology insertion of the environment.

In exploring the feasibility of setting up a software consortium, several issues should be addressed and various tradeoffs should be considered. These fall into the following categories:

- o organizational and legal
- o financial
- o technical
- o marketing

Critics of the consortium concept often raise several objections that we will respond to in this section.

### 7.2.1 Organizational and Legal Issues

The manner in which the consortium is organized is an important issue. Several alternatives are possible. Member companies and/or academic institutions can share equally in the consortium programs or participate only partially in specific areas of interest (and benefit partially in those areas only). Participants can provide people on assignments for a specific length of time (as in the SRC development program) or the consortium can hire its own staff. A combination of both is also feasible in which there is a core staff with member companies also providing people on assignment. These individuals could then return to their parent company and help mature and transition this technology being developed in the consortium to their parent company and get it more widely used. Individuals from academic institutions could then have "industrial experience" while participating in the consortium

and return to the university to familarize students with the software development environment being produced by the consortium.

A potential objection to the consortium alternative is that consortia have typically been staffed by individuals who lack expertise in actually building and delivering MCCR systems. Staffing a consortium with experienced, applications oriented people may not be feasible for a number of reasons. They command higher salaries than a consortium may be willing or able to pay. This is partly because they are in high demand by industry who uses them to bid and win new contracts. It is also possible that such people may not want to work in a consortium environment, being oriented to a business (profit/loss) environment.

An important legal objection is that of anti-trust violation. Most of the U.S. consortia are organized as not-for-profit corporations. MCC was the first to be allowed by the Justice Department to be set up as a for-profit corporation by its shareholder companies. Recent efforts have been under way to remove the barrier of antitrust violation so that consortia can be set up more easily. In fact, there are currently several proposed consortia undergoing Justice Department review.

### 7.2.2. Financial Issues

There are several issues of concern in the financial area for a software consortium:

- o finding a favorable tax climate
- o seeking donations of land and/or equipment to get the consortium going
- o handling joint funding.

Most states in the U.S. are actively seeking growth in high technology areas and are beginning to pass very favorable tax

legislation to encourage high technology joint ventures and consortia. In some instances, donations of state-owned land are a definite inducement to locate in a particular location. Equipment grants are also becoming popular and many of the computer manufacturers would be eager to donate equipment to a software consortium to produce a software development environment for building DoD mission critical systems.

The manner in which joint funding of the consortium is handled is another issue. If the consortium participants include major Defense contractors, small businesses, computer manufacturers, software houses, as well as academic institutions, how is funding handled? Do they all fund it equally and get an "equal" share of the benefits? Should there be government funding involved, and if so, what share? MCC and SRC have resolved this issue in different ways.

### 7.2.3 Technical Issues

There are several technical issues that need to be addressed in a software consortium:

- o Handling of proprietary rights and the rights in data issue
- o Competitive advantage issue
- o Technical approach to building software development environment
- o Responsibility for the resulting product.

It is feasible to protect proprietary rights and still participate in a consortium to build a software development environment for DoD mission critical systems. It is also possible that a member company can have greater benefit from sharing its proprietary idea and making it available to the Government (on some type of mutually beneficial arrangement) or other consortium participants than by keeping it proprietary.

Critics argue against consortia because they feel that such enterprises interfere with their competitive advantage. However, a significant amount of R&D can be done in a pre-competitive posture that would benefit all the participants before they are even competing with each other. One approach to this problem is taken by MCC. MCC gives shareholder companies who participated in the research program exclusive access to an innovation for three years. Pre-competitive research is the focus of the ESPRIT project.

The technical approach that would be pursued in building the software development environment is another major issue. Should an evolutionary approach be taken or a revolutionary one? In fact, should several "competing" approaches go through a thorough design phase and then a choice be made on which one(s) to implement and experiment with in a testbed environment? The level of funding for the consortium would certainly have an impact on these decisions as well as the membership make-up and expertise of the consortium.

Another objection to consortia is that it is not always clear who has responsibility for the resulting product. Since the software development environment is a joint activity and is aimed at the development of DoD mission critical systems, what if something goes wrong? Who is responsible for the project and its results--the consortium itself or the member companies and universities? Who is responsible for product delivery, as necessary, to the SEI and DoD? Who is responsible for marketing, distribution, and maintenance? A partial solution is that since the rights in data issue is under revision, the development environment may not necessarily be required as part of the delivered mission critical system. It could, however, be available to the member companies (perhaps for further productization) as well as the SEI.

### 7.2.4 Marketing Issues

Another issue to be addressed is: what is the market for the software development environment that the consortium would be building? Although the goal is to use the environment to develop DoD mission critical systems, is the market DoD, Defense contractors, the SEI? In fact, do they all form the market of the consortium's environment, and if so, what are their different expectations and needs? This is an issue that can be resolved in a variety of ways. The SEI can be the primary market, or the member companies can be the primary market. Another alternative is that the consortium can "productize" the environment and then make it widely available for DoD use. This issue is also intertwined with several of the organizational and technical issues discussed in the previous sections.

## 7.3 Summary

This section of the report has attempted to define a consortium and give several examples of consortia that exist today as well as a brief description of their operation. Various issues and tradeoffs that must be considered in such an undertaking were briefly addressed. In conclusion, the concept of a software consortium is an alternative for building a software development environment for DoD mission-critical systems and, thereby, significantly improving the current state of software practice for those systems. It is, of course, dependent on the resolution of those important issues for the success of the particular undertaking.

**Time:** The consortium option provides both mid-term and long-range solutions depending on the approach chosen to build the environment. An incremental evolutionary approach would result in products in the mid-term while a revolutionary approach would result in products in the long-term.

**Risk:** The technical risk in this option can span the spectrum from low to high. An evolutionary approach that incrementally builds the environment could control risk easily and result in a low-risk solution. A more revolutionary approach, for example, adopting an alternative paradigm radically different from the current software development paradigm, could have high technical risk but high payoff by resulting in an environment for the long-term.

**Design Method:** The consortium option would develop the environment using a top-down approach.

**Funding Source:** The source of funds for the consortium can be either private or a combination of private and Government. If the funding includes Government funds, their investment would probably leverage the R&D investment made by industry participants in the consortium.

**Implementors:** The developers and implementors would come from industry and academia. In the alternative in which there is some Government funding, Government implementors may be part of the consortium.

**Ownership:** The ownership of the environments would reside in the private sector and therefore be proprietary.

**Business Model:** The consortium option assumes a standard model of commercial business practices in which the technology is licensed to users.

**MCCR Acquisition Stragegy:** Since the environment ownership is in the private sector, developers of MCCR applications desiring to use the environment developed by the consortium could arrange (e.g. by licensing) for use of that environment. The Government would probably not GFE the environment since it is not Government-owned.

**Maintainer:** Two approaches to maintaining the environment are possible: developers or SEI. The maintenance and support by the developer is the likely choice although it may also be possible to have the SEI carry out that function.

**Management Approach:** The management approach is centralized in the consortium.

# REFERENCES

## REFERENCES


1. S. Redwine, et.al., "DoD-Related Software Technology Requirements, Practices, and Prospects for the Future," IDA Paper P-1788, 1984, in press.

2. "A Software Engineering Environment for the Navy," Report of the NAVMAT Software Engineering Environment Working Group, March 31, 1982.

3. "Report of Findings and Recommendations--Software Engineering Institute Study Panel," (N. Eastman, chairman) Institute for Defense Analyses, Record Document D-49, December, 1983.

4. R. J. Hermann, "USDRE Independent Review of DoD Laboratories," Report Prepared for the Undersecretary of Defense Research and Engineering, March 1982.

5. "The Navy/SEI Interface," a briefing by NAVMAT/MAT08Y.

6. Naval Air Development Center, CMS-2 FASP Users' Manual, Revision 6.4, 1 February 1984.

7. "Software Engineering Automation for Tactical Embedded Computer Systems," NOSC, August, 1983.

8. "Ada Programming Support Environment (APSE) Evaluation and Validation (E&V) Workshop Report," Institute for Defense Analyses, 1984, in press.

9. R. A. DeMillo and R. J. Martin, "The Software Test and Evaluation Project, Phases I and II" (6 volumes), Director Defense Test and Evaluation, USDRE, August 1983.

10. "Report of the Rights in Data Technical Working Group," (RDTWG), Institute for Defense Analyses, Record Document D-52, 2 Volumes, January 1984.

11. Software Rights in Data Task Force, "Proposed Reform of Government Rights in Data Clauses," M. Greenberger, et.al., May 1984.

12. DoD STARS Program, "Plan of Action and Milestones for Definition and Preliminary Design of a Joint Services Software Engineering Environment (JSSEE), January 1984.

13. S. Redwine, "The Future Government and Industry Software Tools Marketplace," Proceedings of the 1st Annual Washington Ada Symposium, sponsored by ACM Washington Chapter, Ada Technical Committee (DCAdaTec) and The Johns Hopkins University Applied Physics Laboratory Computer Society, 1984.

DISTRIBUTION LIST FOR PAPER P-1789

CPT David Boslaugh
2221 Jefferson Davis Highway, Rm. #944
Arlington, VA 22202

Paul Clements
Naval Research Laboratory
Code 7595
Computer Science and Systems Branch
Washington, DC 20375

Charles Colello
Plans and Programs Division
Rm 1D679, Pentagon
Washington, DC 20310

LTC Harrington
HQ AFLC/MMEC
Wright Patterson AFB, Ohio 45433

Jim Hess
DARCOM
9N23 AMC
5001 Eisenhower Ave.
Alexandria, VA 22333

John Leary
STARS Joint Program Office
400 Army Navy Drive, 9th Floor
Arlington, VA 22202

Edward Lieblein
OUSDRE/R&AT (CSS)
400 Army Navy Dr., 9th Floor
Arlington, VA 22202

LTC Vance Mall
OUSDRE/CSS
400 Army Navy Drive, 9th Floor
Arlington, VA 22202

Robert F. Mathis (25 copies)
Director, AJPO
400 Army Navy Drive, 9th Floor
Arlington, VA 22202

Carol Morgan
400 Army Navy Drive, 9th Floor
Arlington, VA 22202

LTC Mote
HQ Air Force Systems Command
Office Code ALR
Bldg. 1535 Rm. EE205
Andrews AFB, MD 20334

COL Ken Nidiffer
HQ Air Force Systems Command
Office Code ALR
Bldg. 1535 Rm. EE205
Andrews AFB, MD 20334

Jim Riley
HQ AFSC/DIA
Andrews AFB, MD 20334

Dick Stanley
STARS Joint Program Office
400 Army Navy Drive, 9th Floor
Arlington, VA 22202

Hank Stuebing
Code 50C
NAVAIR DEVCEN
Warminster, PA 18974

David Weiss
Naval Research Lab
Code 7592
Computer Science and Systems Branch
Washington, DC 20375

Other

Dr. Dan Alpert
Director, Center for Advanced Study
University of Illinois
912 W. Illinois Street
Urbana, IL 61801

Betsy Bailey
400 N. Cherry Street
Falls Church, VA 22046

John Bailey
400 N. Cherry Street
Falls Church, VA 22046

Barry Boehm
TRW Defense Systems Group
MS R2-1076
One Space Park
Redondo Beach, CA 90278

Bill Carlson
Intermetrics
4733 Bethesda Avenue, Suite 415
Bethesda, MD 20814

Ruth Davis
The Pymatuning Group, Inc.
2000 L St., N.W., Suite 702
Washington, DC 20036

Richard DeMillo
Georgia Institute of Technology
School of Inf. and Computer Science
Atlanta, GA 30332

Larry E. Druffel
Rational Machines
1501 Salado Drive
Mountain View, CA 94043

Mr. Neil Eastman
Manager, Software Engineering and Technology
IBM Federal Systems Division
6600 Rockledge Drive
Bethesda, MD 20817

Frank McGarry
NASA/GSFC
Code 582
Greenbelt, MD 20771

John Manley
Computing Technology Transition, Inc.
82 Concord Drive
Madison, CT 06443

Ann Marmor-Squires
TRW
Software Development Lab
2751 Prosperity Ave.
Fairfax, VA 22031

Ronnie J. Martin
School of Information & Computer Science
Georgia Institute of Technology
Atlanta, GA 30332

Don Philpot
Software Engineering Technology Corporation
197 Montgomery Rd. MC3
Altamonte Springs, FL 32714

Defense Technical Information Center - (12 copies)
Cameron Station
Alexandria, VA 22314

William Riddle
Software Design and Analysis
1670 Bear Mountain Dr.
Boulder, CO 80303

DoD-IDA Management Office
1801 N. Beauregard St.
Alexandria, VA 22311

IDA

Ms. Louise Becker
Mr. Matthew Berler
Mr. J. Frank Campbell
Dr. Jack Kramer
Ms. Sarah Nash
Dr. Thomas H. Probert
Mr. Samuel T. Redwine, Jr.
Mr. John Salasin
Dr. Marko M. Slusarczuk
Mr. E. Ronald Weiner
Ms. Carol Powell - (2 copies)


DATE
ILME