18th Annual Software Enginnering Workshop
Lessons Learned Applying CASE Methods/Tools To Ada Software Development Projects

December 1, 1993

Maurice H. Blumberg
(301)240-6018
blumberm@wmavm7.vnet.ibm.com
Dr. Richard L. Randall
(719)554-6597
randallr@wmavm7.vnet.ibm.com

STARS Project
IBM Federal Systems Company
800 N. Frederick Ave.
Gaithersburg, Md. 20879

Abstract

This paper describes the lessons learned from introducing CASE methods/tools into organizations and applying them to actual Ada software development projects. This paper will be useful to any organization planning to introduce a software engineering environment (SEE) or evolving an existing one. It contains management level lessons learned, as well as lessons learned in using specific SEE tools/methods. The experiences presented are from Alpha Test projects established under the the STARS (Software Technology for Adaptable and Reliable Systems) project. They reflect the frontend efforts by those projects to understand the tools/methods, initial experiences in their introduction and use, and later experiences in the use of specific tools/methods and the introduction of new ones.

Preface

This paper was prepared by the Maurice H. Blumberg and Dr. Richard L. L. Randall of the IBM Federal Systems Company, located at 800 North Frederick Avenue, Gaithersburg, MD 20879. Many thanks to fellow IBMers, Terry Snyder and Ron Backus of the GPS project and Frank Hyson and Neal Walters of the ACE project, for their inputs on lessons learned from their projects.

The following trademarks are used in this paper.

AIX, RISC System/6000, PS/2, and IBM are trademarks of the International Business Machines Corporation.

Rational, R1000, R300C, and Rational Environment are trademarks of Rational Corporation.

AdaMAT is a trademark of Dynamics Research Corporation.

Interleaf is a trademark of Interleaf, Inc.

Teamwork is a trademarks of Cadre Technologies Inc.

DocEXPRESS, DoDEXPRESS, and Methods_Help are trademarks of ATA Inc.

ObjectMaker (Adagen) is a trademark of Mark V Ltd.

STATEMATE is a trademark of i-Logix, Inc.

RTrace is a trademark of Protocol Company.

PVCS is a trademark of INTERSOLV, Inc.

CCC is a trademark of Softool, Inc.

LOGISCOPE is a trademark of Verilog, Inc.

Windows is a trademark of Microsoft Corporation.

Vax is a trademark of Digital Equipment Corporation.

Keywords: Software Engineering Environment, CASE, STARS, Methods, and Lessons Learned.

Table of Contents

Introduction

The objective of the Software Technology for Adaptable, Reliable Systems (STARS) Program is to develop, engineer, and integrate technologies that, when employed in the development of DoD software systems, will improve quality and predictability, and reduce the cost of development. STARS believes these improvements will primarily result from applying the "megaprogramming" paradigm which involves designing and building systems based upon tailorable reusable components, improvements in the software process, and through technology support for the development process. The STARS solution will embody these concepts with megaprogramming processes supported by software engineering environments (SEE).

STARS will accelerate a transition to a megaprogramming paradigm by demonstrating the benefits of megaprogramming on real DoD projects. This is being accomplished by a STARS Demonstration Activity, which was initiated in the 3Q92 and involves multiple "demonstration projects" in different application domains. The demonstration projects will formally begin their performance phase in 4Q93.

The STARS program is evolving and instantiating SEE solutions to support the demonstration projects. Prior to 4Q93, the STARS SEE solutions will evolve from the 1991-1992 "alpha versions" to integrated versions in 4Q93, running on various hardware platforms. In 1991 the IBM STARS team defined an initial generic instantiation of a SEE solution, designated as the IBM STARS Alpha SEE.

During 1Q91, IBM created the organization and structure to support "Alpha Test" projects, developing Ada software. The purpose of Alpha Test projects is to:

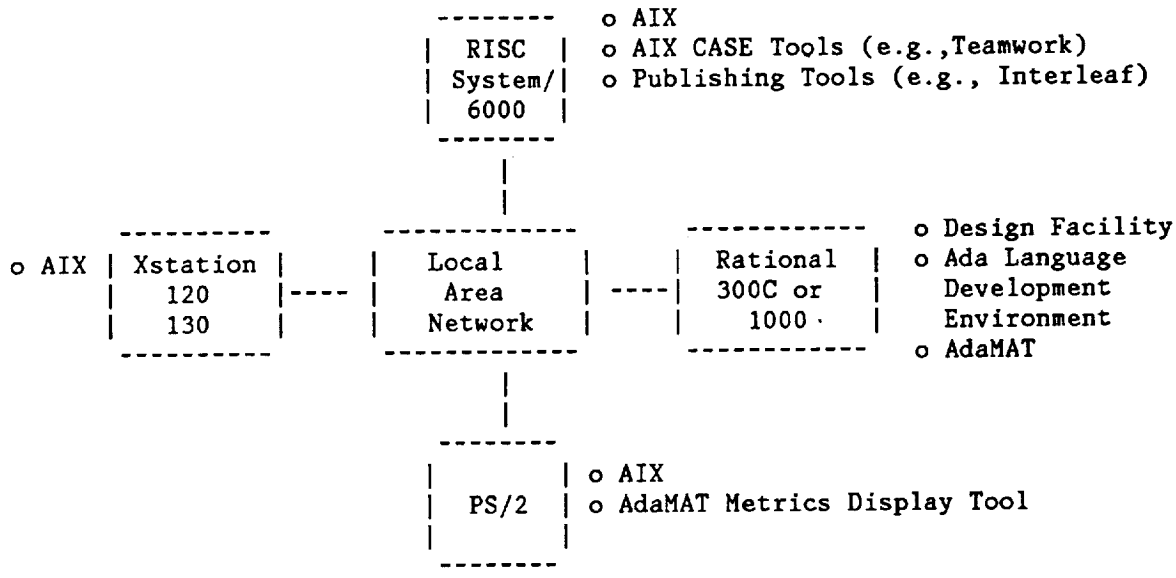
- Gain early experience and feedback in the use of the IBM STARS Alpha SEE Solutions
- Provide vehicle for early technology transfer of IBM STARS capabilities
- Be a precursor for STARS Demonstration Projects in defining:
  - A technology transfer process
  - How to support projects using a SEE
  - How to capture lessons learned information

IBM STARS Alpha SEE Solution

The IBM STARS Software Engineering Environment (SEE) is a combination of hardware platforms and software tools which support Ada software development from requirements analysis through code generation, testing and maintenance. The SEE is adaptable, i.e., it is tailorable to a "SEE Solution" which meets the specific needs of a project.

The IBM STARS Alpha SEE is based on IBM's AIX CASE Solutions. These solutions consist of IBM and IBM Business Partner products that support the software development process through software engineering methodologies, distributed workstation-based environment, and open system applications. The AIX CASE Solutions provide an open framework and a set of solutions and products supported across the range of the RISC System/6000 family. The IBM STARS team incorporated value add efforts from STARS into solutions where applicable.

The initial IBM STARS Alpha SEE solution was assembled from IBM AIX CASE Business Partners and other AIX CASE vendors. The figure below depicts the major hardware and software components.



The software tools are integrated at varying levels within the initial solutions, e.g., there is a Rational/Teamwork interface which allows Teamwork analysis diagrams and text to be imported into Rational and a Teamwork/Interleaf interface for generating specifications and design documents.

The IBM STARS Team supported the Alpha Test projects in modifying the solution and adapting it to fit the project's process and methods.

IBM STARS Alpha Test Projects

The IBM STARS team has established three Alpha Test projects which are using the capabilities of the IBM STARS Alpha SEE to develop Ada software. The IBM STARS team has provided support to these Alpha Test projects and has collected feedback from them on their experiences with their SEE solutions.

The current IBM STARS Alpha Test projects are as follows:

- Global Positioning System (GPS)
- Ada CASE Engineering (ACE)
- Forward Area Air Defense (FAAD) Electronic Support Measure (ESM) Non-cooperative Target Recognition (NCTR) System (FAADS)

The table below provides a summary of the major tools used by the Alpha Test projects, categorized by system life cycle activities (see Appendix A for a description of these tools).

Life Cycle Activity	GPS	ACE	FAADS
Analysis	Teamwork	Teamwork STATEMATE	N/A
Design	Adagen Rational	Adagen Rational	Teamwork/Ada & DSE
Implementation	Rational	Rational	TLD Ada Compiler
Document Generation	Teamwork DocEXPRESS Interleaf Rational	Rational	Teamwork DocEXPRESS Interleaf
Reverse Engineering	Adagen	Adagen	Adagen
Requirements Traceability	Rational	Rqt RTrace RTM Rational	Manual

Table 1. Alpha Test Projects SEE Tool Usage

The Alpha Test projects are using a wide range of SEE tools and methods covering the entire system life cycle. However, to provide some focus to the Alpha Test efforts, each project was asked to concentrate on a specific aspect of the lifecycle. GPS is focusing primarily on system engineering and software design, ACE on requirements traceability and software design, and FAADS on software design and reusability.

Each of the Alpha Test projects are described in the sections that follow. The descriptions include their hardware/software configuration, tool usage, and lessons learned from introducing and using CASE tools/methods. The "introduction" lessons learned reflect, for the most part, the frontend efforts to understand the tools/methods and some early experiences in the use of them. The "using" lessons learned reflect, for the most part, later experiences in the use of the specific tools/methods and the introduction of new ones.

Lessons Learned Introducing and Using CASE Tools

Global Positioning System (GPS)

Description

The Global Positioning System is a space based navigation system consisting of a constellation of Space Vehicles (SVs) and a ground support system. The GPS project is responsible for the hardware and software development of the ground support system. This includes software to generate the navigation data, upload the SVs, process telemetry data, and in general, provide commanding and control of the SVs. Project responsibilities also include the maintenance and upgrade of hardware at the remote tracking stations and master control station. The GPS project is in its 13th year of development and follow-on contracts. The system consists of approximately 1 million SLOC, mostly in JOVIAL.

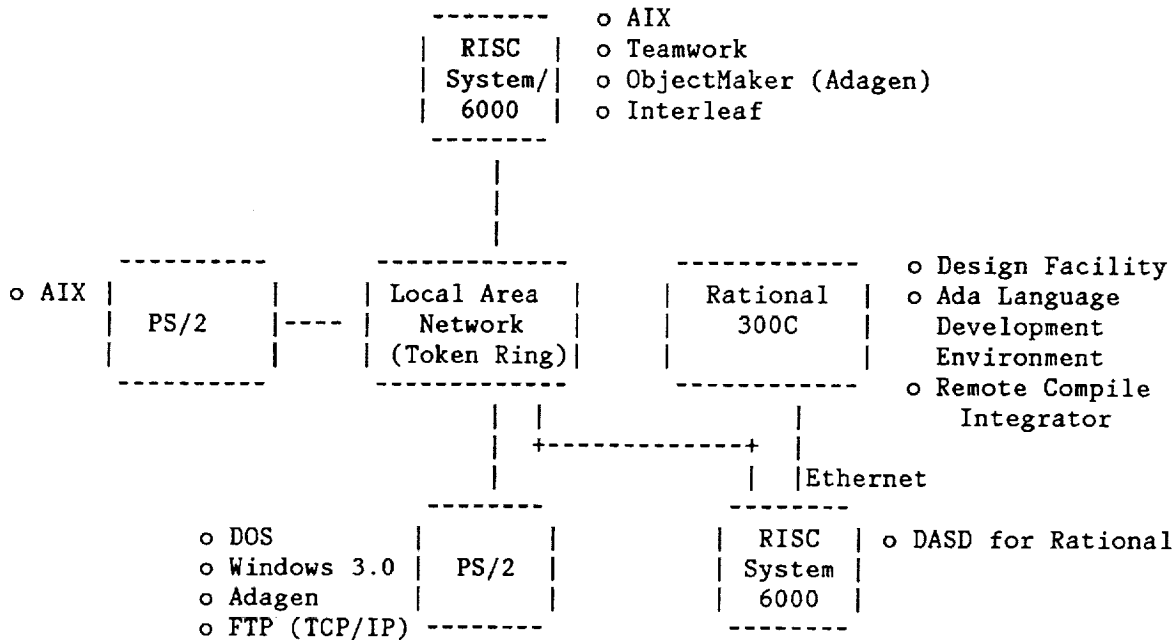
The current GPS effort includes the development of two new Computer Software Configuration Items (CSCIs), requiring approximately 32 KSLOC of Ada. These CSCIs are being developed on RISC System/6000 and workstations; they will also be run operationally on RISC System/6000 and Rational hardware.

Initially, the IBM STARS Alpha SEE was used to support the Ada software design, Software Design Document (SDD) generation, and Ada code development for the CSCIs. Subsequently, the SEE was also used to develop a Software Requirements Specification (SRS) using OOA and the SEE Tools, Teamwork and Interleaf. Future plans include use of Teamwork/Ada for high-level software design.

GPS Hardware/Software Configuration

The GPS hardware/software configuration is depicted in the figure below.

GPS Hardware/Software Configuration



GPS SEE Tool Usage

The IBM STARS Alpha SEE tools currently being used by GPS are Teamwork (and DocEXPRESS), ObjectMaker (formerly known as Adagen), and Rational. The use of these tools is described below.

Teamwork

GPS initiated an effort to use OOA methods and Teamwork to generate an SRS for the Onboard Processor CSCI (Computer Software Configuration Item). Because of funding and scheduling issues, the system engineer (SE) responsible for the SRS could not take any formal classes on OOA or Teamwork. This provided an opportunity to determine the effectiveness of learning a CASE tool and the methods it supports without formal classroom training. The training approach developed by STARS for the GPS system engineer was as follows:

- Go thru on-line Teamwork/SA tutorial - 1 day
- Go thru "Strategies for Real Time Systems Specification" book by Derek Hatley - 3 days
- Go thru "Object-Oriented Analysis: Modeling the World in Data" and "Object Lifecycles: Modeling the World in States" books by Sally Shlaer and Steve Mellor - 3 days
- Go thru on-line training tools, DoDEXPRESS and Methods_Help - 1 1/2 days
- Experiment with Teamwork - 2 days
- Bring in Cadre system engineer (no charge) for a Q & A session - 1 day

GPS also used the DocEXPRESS tool which generates a DoD-STD-2167A compliant SRS from the Teamwork developed OOA model. The DocEXPRESS vendor, ATA Inc., was funded to enhance the tool to generate requirements traceability matrices as part of the SRS.

Teamwork is also being considered by the GPS software engineering team to support an object based Ada software engineering methodology. This includes use of Teamwork/Ada for creating Ada Structure Graphs as well as compilation dependency and Buhr diagrams. The interface between Teamwork and Rational is also being studied to determine how far to go with Teamwork before migrating to the Rational.

ObjectMaker (Adagen)

ObjectMaker (Adagen) was originally used by GPS for early conception of design, preparation of high level architecture and design overview material. Later on, it was used for reverse engineering of Ada code to diagrams for inclusion in the SDD (Software Design Document). Specifically, for conceptual design, Adagen was used to draw bubble charts which showed relationships between objects and message flow. In addition, Booch class category charts were used for CSC (Computer Software Component) evolution, and Buhr diagrams were used for CSU (Computer Software Unit) evolution and interaction. Reverse engineering of Ada code, using Adagen, was employed to ensure that the Ada graphical diagrams in the SDD were consistent with the Ada source code. The diagrams generated by Adagen were compilation dependency diagrams, which showed the "withing" relationship between Ada packages and Buhr diagrams, which graphically depicted the contents of each Ada package.

Rational

The Rational design facility and environment was originally used by GPS for development of PDL for preliminary and detailed design, generation of SDDs, code development, some unit testing, and requirements traceability to code. The Rational Design Facility was further customized to provide the capability to include Ada package specifications in-line in the Software Design Document. Additionally it provided for Appendices to address what Non-Developed Software (NDS) is being used in the system. In addition, the newly available Remote Compilation Integrator (RCI) was used to allow source code on the Rational to be compiled on the RISC System/6000. The RCI was used in conjunction with Rational's Configuration Management Version Control.

GPS Lessons Learned from Introducing CASE Tools

The lessons learned by the GPS project from introducing the IBM STARS Alpha SEE are described below:

- Significant start up preparation and cost for a new Ada project

  For GPS, an existing project transitioning to Ada, many new things needed to be learned by the software developers, including a new language, a new set of tools and methods, and a new process (i.e., DoD-STD-2167A). This required considerable training costs and a significant learning curve for the project team.

- Customization of SEE tools requires significant resources

  Several of the tools used by GPS required customization, e.g., Rational needed customization to produce an SDD which conformed to the customers requirements, provide requirements traceability, and imbed diagrams from Adagen. Additional customization of Rational was required to extract PDL for the SDD. This customization required several labor months of effort, with additional customization still required.

- Choose a project methodology and train developers early

  The decision to use Ada as the programmming language was made well after the start of the current GPS effort. Thus, choice of a method and tools, and all the startup costs mentioned above, were not part of the initial project planning and required significant adjustments to the original plans and schedules.

- Have engineers use object oriented analysis for specifications

The GPS system engineers used functional decomposition methods to generate their SRS, while the software engineers used object-based design methods for the Ada software design. The use of an object oriented approach to defining requirements would reduce the effort required to transition between the specification and design phases.

- Use Ada as the design language

  Using Ada as the design language provides a compilable design which can be checked for completeness/consistency of interfaces, data definitions, etc.

- Preserve ability to extract PDL after code completion

  The ability to extract PDL from the code (e.g., for inclusion in an SDD) allows design and code to be maintained in a single place and decreases configuration management requirements.

- Agree with customer on diagramming and PDL techniques early into a project

  As mentioned above, choice of Ada and associated design methods and tools were made after the start of the current GPS effort. As a result, several iterations with the customer were required to gain agreement on issues regarding the formats and styles of Ada diagrams (e.g., Buhr diagrams) and PDL.

- Plan for a target code version and unit tests on target

  The GPS Ada source code was developed, compiled, and partially unit tested on the Rational hardware. Since the target machine is the RISC System/6000, provision was made for comprehensive unit testing on the target machine, even though the Ada code is "compatible". In addition, configuration management of source and target software needed to be provided for.

- Need additional personnel roles

  New project roles are required as a result of introducing SEE methods and tools into a project. In addition to a methods/tools "guru(s)" for consultation, GPS required the following additional personnel roles:

  - Rational System Administrator
  - Rational CMVC/RCF Administrator
  - Rational Design Facility Customizer
  - Adagen Support Expert

  Note: Multiple roles can be played by one person.

GPS Ada Lessons Learned from Using CASE Tools

The lessons learned on the GPS project from using the IBM STARS Alpha SEE tools/methods are described below:

Teamwork Lessons Learned

- An understanding of the basic capabilities of Teamwork was gained without formal classroom training.

  The Teamwork tutorial which provides hands-on training is adequate for new users to learn the basic capabilities of Teamwork. Building data flow diagrams, entity relationship diagrams, and defining entities, data flows, processes, stores, their corresponding attributes is relatively straight-forward. Teamwork/Ada was also fairly easy to use without formal training. The various user's guides, provided for each of the editors, are also well-written and provide detailed information on the use of the editors.

- Formal classroom training is required for understanding object-oriented methods.

Learning any new methodology is a challenge. However, object-oriented methods require a totally different approach and way of thinking that is quite different from traditional structured analysis (SA) and design methods. For example, most system engineers are so ingrained in using some form of SA for requirements analysis that without formal training, they will have a difficult time understanding OOA and transitioning from SA to OOA methods.

- Generating an SRS which satisfies the customer's DoD-STD-2167A DIDs (Data Item Descriptions) requires considerable tailoring of the Teamwork provided templates.

  The Teamwork document generation capability is powerful, but fairly complex. Tailoring the SRS and extracting the specific data requires significant time and effort. Also, the lack of a table building capability contributes to the difficulty in generating documents. The use of DocEXPRESS simplified generation of 2167A compliant documentation from Teamwork, but it required considerable enhancements (by the DocEXPRESS vendor) to generate an SRS which satisfied the customer's DoD-STD-2167A DIDs (e.g., provide requirements traceability matrices).

Adagen Lessons Learned

- An understanding of the capabilities of Adagen was gained without formal classroom training.

  Adagen was fairly easy to learn and has a well-designed user interface. The Adagen tutorial provides hands-on training and is adequate for new users to learn the basic capabilities.

- Reverse engineering of Ada code to create design diagrams for documentation and communication of design is very effective.

  The reverse engineering of Ada code ensured that the Ada graphical diagrams in the SDD were consistent with the Ada source code. However, some manual editing of the diagrams generated by Adagen was required.

Rational Lessons Learned

- Expense and overhead of supporting the Rational development environment is high.

  The Rational development environment has served the GPS project well in developing and testing of Ada code. However, Rational is somewhat inflexible in its ability to increase the number of users. Tokens are also fairly expensive and cannot be leased. Upgrading the Rational hardware to support additional users can also become an issue.

- A significant effort was required to customize the Rational Design Facility (RDF).

  The customization of the RDF to include Ada package specifications in-line in the Software Design Document and provide appendices for NDS required several labor-months to complete.

- Extensive training was required to fully utilize Remote Compilation Integrator.

  Extensive training was required to fully understand the implications of using the Remote Compilation Integrator, in conjunction with Rational's Configuration Management Version Control.

- Unit testing within Rational of Ada code using multiple COTS tools requires a significant effort.

  The Rational debugger works well for code which is machine independent and does not call the operating system or other COTS software. Unfortunately, the GPS application is dependent on the operating system and on a COTS GUI package. The Remote Procedure Call (RPC) capability of Rational would enable GPS to use the debugger through more of the unit testing cycle. However, a significant amount of resources is required to implement and support RPC.

Integration Lessons Learned

- Integrating tools to build an environment to support the entire life cycle is difficult.

  Although each of the GPS CASE tools performed well in its intended phase of the life cycle, their underlying data representations are not easily shared across other tools or products. Incorporating changes in one phase into the products of a previous or future phase is difficult if another tool is required to produce those products.

- No CASE tool currently addresses the need for page integrity.

  As software is modified to incorporate enhancements, the supporting document's page numbers cannot change. Add pages are used to accomplish this. Many CASE tools can generate a document but they are not good publishing tools. As a result, none support page integrity. It is hoped that Interleaf, which is a publishing tool, will address this issue. Unfortunately, using another tool adds to the problem of configuration management across tools.

Ada CASE Engineering (ACE)

Description

The Ada CASE Engineering (ACE) project was established at the end of 1988 to perform ongoing evaluations of tools and methods that can improve the process of developing Ada software, from proposal activity through maintenance. To accomplish this, a CASE tool environment laboratory was set up at FSC Manassas and investigations of various methods which could be successfully used with these tools were performed (this included conducting pilot projects).

Since 1988, the ACE project has played a lead role in infusing new tools and methods into FSC systems engineering and software development, with the goal of improving the productivity and quality of Ada software. This has included providing a variety of training classes for both tools and methods. ACE has also supported new Ada projects that use CASE tools and the Rational Ada development environment.

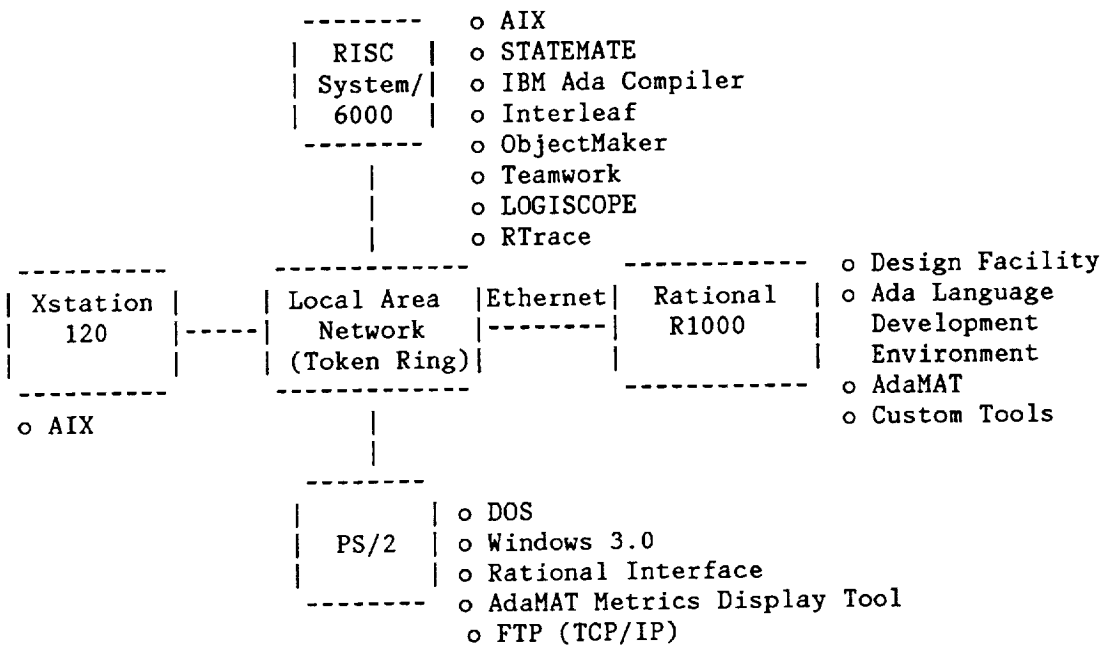
In 1992 the ACE project has continued to investigate the application of CASE tools and methods for Ada systems development. Early this year the project supported the IBM Manassas' efforts to evaluate the Integrated CASE (I-CASE) RFP from the U. S. Air Force. The I-CASE RFP was reviewed and found to contain more than 900 requirements. The large number of requirements gave ACE an opportunity to put the draft RFP under the RTrace requirements tool so that they could better manage the total scope of the requirements.

The ACE project continues to strongly focus on CASE tools hosted on the IBM RISC System/6000. A major emphasis has been on the requirements definition and modeling tool, STATEMATE, from i-Logix, the requirements traceability tool, RTrace, from Protocol as well as other tools. Efforts are on-going for investigating reuse techniques including the formulation of an object-oriented domain model for an existing project. ACE is using STATEMATE, RTrace, ObjectMaker, Rational and other tools to develop a SEE.

ACE Hardware/Software Configuration

The ACE hardware/software configuration is depicted in the figure below.

ACE Hardware/Software Configuration



The specific components of configuration are as follows:

- RISC System/6000 Model 530
- Rational R1000, Series 400
- Token Ring and Ethernet LAN
- Xstation terminals for LAN access
- PostScript Printer (IBM 4216)
- STATEMATE, ObjectMaker, Teamwork, RTrace, Ada Compiler, LOGISCOPE, and Interleaf on RISC System/6000
- Rational Design Facility (RDF) on Rational R1000
- Rational interface, AdaMAT Metrics Display Tool (MDT) on PS/2

ACE Tool Usage

The IBM STARS Alpha SEE tools currently being used by ACE are STATEMATE, RTrace, Rational, and ObjectMaker. Other tools used include LOGISCOPE, AdaMAT and the AdaMAT Metrics Display Tool (MDT), text editors, and text postprocessors. The use of these tools are described below.

STATEMATE

STATEMATE, from i-Logix Company, is a graphic modeling tool. It ties together three types of diagrams which can be used to model systems. The Statechart is very much like a state transition diagram. The Activity chart is like a data flow diagram and Module charts show the structural view of a system. The "languages" of the Statechart and the Activity chart are very sophisticated, including time based functions, yet simple to draw and manipulate. STATEMATE includes extensive model checking, DoD documentation generation, model simulation and prototype code

generation. The prototype code generated by STATEMATE can be used to drive screen panels, and take panel actions as inputs.

STATEMATE is used by ACE to perform requirements analysis and modeling of the problem domain. This methodology uses a technique for characterizing requirements as objects and accurately models system behavior. STATEMATE performs completeness/consistency checking and captures all definitions of external and internal data. STATEMATE produces the SRS and IRS (Interface Requirements Specification) documents according to DoD-STD-2167A standards. The ACE project customized this documentation facility to better match the object-oriented methodology that it is defining as part of its SEE development effort.

Modeling

The ACE group used STATEMATE to build an object-oriented analysis/design model. The model is built using Module Charts to represent the objects, Statecharts to represent the behavior of each object and the interaction between objects, and an Activity Chart to show the context of the problem. The Statecharts were executed to illustrate performance modeling (run-time overhead and processing times), event-response scenarios and error recovery scenarios. The STATEMATE model and timing chart outputs will be documented in Version 2.1 of the model software specification that the project is writing.

Rapid Prototyping in Ada

There was a two week effort to evaluate the Ada prototyping capabilities of STATEMATE. A Traffic Light Model and Panel were successfully built and executed. The last obstacle regarding the prototype code driving the display panel was solved, by using the "hooks" option. STATEMATE code generation does not normally generate hooks for states, activities, etc. unless specifically asked for.

The Ada code for the system (1 Statechart, 1 Activity chart, 8 states and one panel) amounted to about 1500 SLOC of Ada. The Ada code generated models the states and events defined in the system, and drives a simple panel.

RTrace

The RTrace requirements traceability tool currently is used by ACE as a stand-alone package to identify requirements from a customer A-Spec and to build a requirements data base. These requirements are then allocated to various system objects and components. This allocation may be used to do impact analysis if, for example, a requirement should change. This tool has extensive reporting capabilities, and some report formats were customized to better support technical and project management activity.

RTRACE is currently hosted on a SUN or DEC platform. Protocol has recently ported RTRACE to the RISC System/6000.

Rational

The Rational design facility and environment is used by ACE to produce various DoD-STD-2167A documents including preliminary SDDs and IDDs from the preliminary design activity, and final SDDs and IDDs (Interface Design Document) from the detailed design activity. Rational's automated document generation facility allows documents to be produced from one common evolving source. An interface from Rational to ObjectMaker is also used to allow diagrams generated by ObjectMaker to be included in the appropriate documents. Rational is used for design, code development, unit testing, and requirements traceability to code.

ObjectMaker

ObjectMaker is used by ACE to support the development of high level graphical Ada design constructs that result from the STATEMATE model. ObjectMaker is then used to develop Ada code design diagrams (Buhr diagrams), and from these diagrams, generate Ada skeletal code. After code completion, ObjectMaker is used to perform reverse engineering to create accurate Ada graphical

diagrams from the Ada code, for reviews and incorporation into the SDD. The diagrams generated by ObjectMaker are compilation dependency diagrams, which show the "withing" relationship between Ada packages and Buhr diagrams, which graphically depict the contents of each Ada package.

LOGISCOPE

LOGISCOPE is used to perform metrics analysis of Ada code to enhance reliability, maintainability, and portability. By quantifying the Ada software quality, LOGISCOPE identifies potential problems in the Ada code, provides test coverage and complexity metrics, and addresses performance issues. LOGISCOPE is also used to provide graphical reports of the metrics.

AdaMAT

AdaMAT is an Ada metrics tool that runs on the Rational. Like LOGISCOPE, it checks on the conformance of Ada code to a wide variety of quality indicators. AdaMAT provides a number of reports and generates data that may be further analyzed off-line with a PS/2 based tool.

ACE Lessons Learned from Introducing CASE Tools

The lessons learned by the ACE project from introducing the IBM STARS Alpha SEE are described below:

- The single most important key to the success of a project is still to understand the problem thoroughly.

  There is still no substitute for sound systems and software engineering. SEE tools and methods only provide support to this process by providing a structured approach to recording and checking the results of sound engineering, and a means for communicating the results more clearly to others.

- Adequate training in tools/methods must be provided.

  SEE tools/methods require significant training to learn. Lack of adequate training can lead to misuse of tools, causing a negative impact on a project, and resulting in tools becoming expensive "shelfware".

- New methods and tools require considerable time to learn and this time must be allocated to a project schedule.

  In addition to proper training, SEE tools/methods require considerable hands on use before developers are proficient in their application to real problems. This learning curve must be accounted for, especially in the front end costs of a project.

- Tools require considerable lead time before they are operational.

  Significant customization of tools to a project's specific needs and documentation of detailed project standards and procedures are required to make SEE tools "operational". In addition, bridges between tools, e.g., Adagen and Rational need to be developed.

- New methods/tools need to have a strong project advocate.

  Because of the significant startup costs mentioned above, and the the long lead times required before new methods/tools begin to make an impact, a strong project advocate is needed to maintain the project commitment until the benefits of the tools/methods are realized.

- A project should have a 'toolsmith' who can customize tools to the project when necessary.

  In addition to the initial customization of tools, there is an ongoing requirement for customization to tailor tools to the changing needs of a project.

- Consider whether a tool/method might not 'scale up' to a large project.

Many tools/methods/notations look very good when applied to relatively simple problems, but yield very complex and difficult to understand results when applied to large, fairly complex problems. In evaluating tools/methods, system developers need to look beyond the simple examples that are used to demonstrate the application of those tools/methods and determine if they will scale up to their specific problem domain.

- Having tools available in the office via networking is a productivity enhancer.

  Having access to SEE tools from office desktop computers, provides convenient access to these tools, while taking advantage of existing computing resources.

- New tools and methods should not be seen as a panacea.

  This is another way of emphasizing that there is still no substitute for sound systems and software engineering.

ACE Lessons Learned from Using CASE Tools

The ACE project has recently been developing a software engineering environment to be used by an upcoming Ada project at Manassas. This tool's environment will provide life-cycle support from requirements tracing and capture through software design and development, testing and maintenance activities. The lessons learned are from the experience in developing this SEE and from other activities.

STATEMATE

- The STATEMATE panel generator and Ada Prototyper provide an interesting and informative view into a model.

  These tools have great potential for modeling network and processor performance, timing, concurrent processing, error paths and event-response. They are also good for demonstrating user interfaces, but the generated code models STATEMATE states/events/etc, and is not likely to be useful for real code design/development.

- The language and semantics of STATEMATE require a steep learning curve.

  STATEMATE is a powerful tool that requires a fair amount of time to fully understand. For example, it was very time consuming to figure out the best way to model the first object under STATEMATE; however, once this was understood it became fairly mechanical to add new objects.

  The ACE estimate for training is a minimum of one week of hands-on training, followed by three weeks of hands-on prototyping with available expert consulting. The training and consulting could be provided internally as sufficient skills are developed and a course/prototyping exercise is developed.

- Definition of STATEMATE naming conventions is very important.

  Good naming conventions are important not only for STATEMATE modeling and prototyping activities, but for all CASE tool efforts. In the STATEMATE object model, each object service needs two conditions and two events associated with it as well as data-items to define simulated processing times. All these items require names. Some of these items need to be global in scope, some are of local scope - they can be defined either way. (In this sense STATEMATE is somewhat like Ada - you can overload a name in the proper context).

RTrace

- RTrace supports 2167A requirements traceability.

  Based on the testing done, the review of the documentation and the support received from Protocol Company, ACE recommended using RTRACE for projects requiring 2167A

traceability. RTrace can also be used for proposal requirements management, e.g., tracing RFP requirements to proposal sections and responsible authors.

- RTrace is easy to use and should not require formal education.

  The current version of RTRACE has a fairly straight-forward and menu oriented, key-intensive user interface. Protocol is planning to have a future Motif user interface. Protocol should help set up the first project's database. It will save a lot of time later if all the "objects" and their relationships are defined completely and correctly before the requirements are loaded. Examples of RTRACE objects are: Configuration Items, Functions and Sub-Functions, responsible engineer, test drop, build number, test procedures, etc.

- RTRACE is not integrated with any other CASE tool.

  Because RTrace is a standalone tool, this means all updates must be done manually. Updating data from tool to tool will be easier in the future windows environment.

- Projects using RTRACE will need a "guru" to support users.

  Projects using RTRACE can use co-ops or junior level people to parse existing documents to enter into the RTRACE database. After that, knowledgeable engineers, programmers, and testers must assign characteristics and link objects to each requirement being traced. A tools "guru" is needed to customize reports and do some tool administration functions.

- The current release of RTRACE does not support any automated configuration management or version control.

  Some form of version control is planned for future releases. This should provide the potential for integration with configuration management tools.

Miscellaneous Lessons Learned

- Many tools need to be customized before they can be used on a project.

  Most CASE tools, such as STATEMATE and Rational, require customizing in order to support the desired methodology and unique requirements of a project (including the customer's requirements). Thus, for most projects a toolsmith is a necessity.

- Many new tools have a significant learning curve.

  If the users do not have adequate learning time and motivation, this will likely kill the tool's chance of acceptance with users and management. If possible, new tools should be chosen to operate in the same manner as earlier tools they replace. This will give the user the sense that their previous skill with the older tool has not been wasted.

- Educating the user group is an important part of introducing a new toolset.

  Once a toolset and methodology are selected, an educational plan (schedules, preparation, funding) needs to be addressed as early as possible that will support this methodology. In the beginning, tools environments may succeed by virtue of attracting enthusiastic individuals. Ultimately a good teaching method is needed to extend tool use to those who would rather keep the status quo.

- Anticipated users of a toolset should have training and access to the toolset prior to its needed use on a project.

  If possible, users should gain familiarity with a toolset before the demands of the project are felt. There should be some opportunity to experiment with the tool on a prototype or small pilot project before using it on the actual project.

- Bringing users onto a technology transition oriented project such as the ACE project, before their real use of the tool is required, eases the learning process and makes the user more receptive.

The ACE project found that training new users by participating in the ACE project was a very effective way of producing enthusiastic users, but is limited in terms of how many users can be trained this way. Another approach is to seek out receptive individuals who express an interest and enthusiasm for working with new tools and methods. Obviously lead positions need to be filled with such persons.

- A course in how to write a good specification can improve the quality of specifications.

  The ACE project developed a 2-hour model SRS writing course that teaches the fundamentals of "good" SRS writing techniques. This course is taught just before the students are to begin developing real SRSs and the techniques are fresh in mind. A class needs to be timed for optimal effectiveness. If it's given too soon, the motivation may not be there, and retention may be a problem. Waiting too late may overload users with too much last minute information. The lesson is to have material on the shelf ready to go at the optimum time. A good teaching technique is to use illustrated examples of the principles being taught. Continuous process improvement and defect analysis techniques should be applied to a course, after student critiques are received, to improve the course for next time.

FAADS

Description

The FAADS contract, which was started on April 1, 1991, is responsible for the development of hardware and software for a passive ESM system to support tactical forward area defensive weapons platforms in detecting airborne threats and cueing weapons operators. Magnavox is the prime contractor and the IBM Federal Sector Company in San Diego is the software developer.

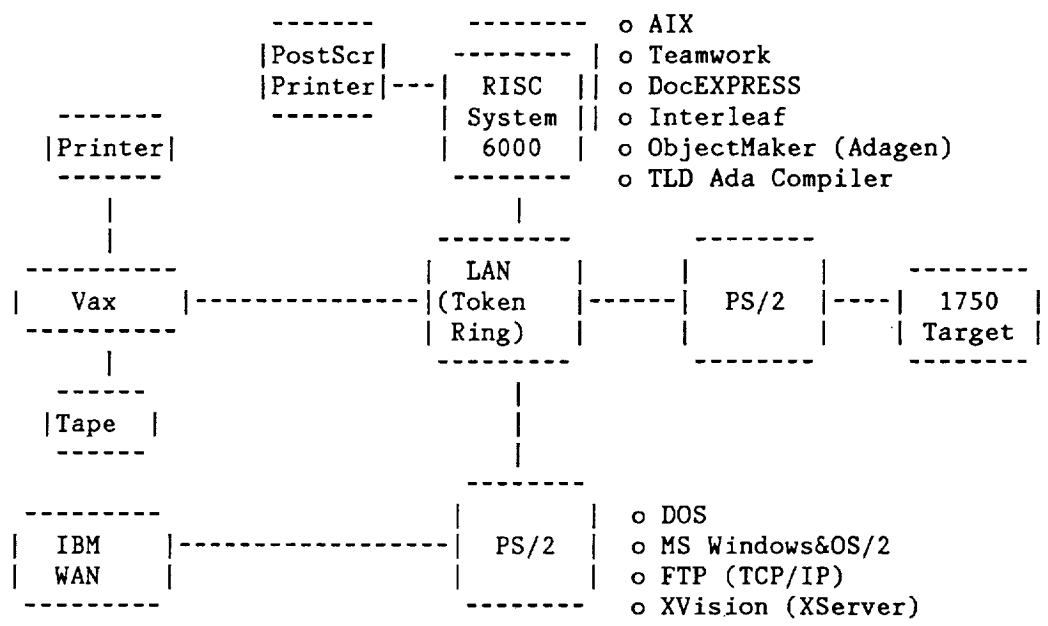
Although this program is informally known as FAADS, it is actually only a portion of a much larger program, the Forward Area Air Defense System. The specific portion under contract is the AN/VSX-2 program, also known as the Non-Cooperative Target Recognition program, or NCTR-1. IBM is under contract with Magnavox Electronic Systems Company to develop a portion of AN/VSX-2.

The FAADS Software will be developed in two phases, with a Model I consisting of one CSCI developed in the first phase and a Model II consisting of three CSCIs developed in the second phase. There is planned heavy reuse of Ada code from model I to Model II. The Model I software architecture is based on an existing Ada system consisting of approximately 15 KSLOC of Ada code. Model I software will consist of approximately 12 KSLOC of Ada code. The software will run on 1750A processors; Ada compilations, which were originally being done on a uVax II, are now being performed on the RISC System/6000.

FAADS Hardware/Software Configuration

The FAADS hardware/software configuration is depicted in the figure below.

FAADS Hardware/Software Configuration



The specific components of configuration are as follows:

- Two RISC System/6000 Model 320s
- Token Ring LAN
- PS/2 Workstations for LAN access
- PostScript Printer
- Teamwork, DocEXPRESS, Interleaf, and Adagen on RISC SYSTEM/6000
- Ada TLD Compiler on RISC SYSTEM/6000 and Vax

FAADS Tool Usage

The STARS Alpha SEE tools currently being used by FAADS are Teamwork, ObjectMaker (Adagen), DocEXPRESS, Interleaf, and PVCS. A member of the FAADS team, who had some prior background in tools and methods, served as a SEE tools/methods consultant.

Teamwork

Originally, IBM planned to use Teamwork/SA for software requirements analysis, but since the prime contractor retained this responsibility, this was not possible. IBM did use Teamwork/SA on a very limited basis to analyze the SSDD (System Segment Design Specification) and the SRS early in the program. This effort yielded some useful feedback to the prime contractor as to omissions and inconsistencies.

In another change to the original plans, IBM elected to use Teamwork/Ada rather than Adagen to support software design, for these reasons:

- Teamwork had built-in multi-user support
- Teamwork had built-in document production support

- Cadre delivered a new Teamwork module called the Ada Design Sensitive Editor (DSE) which was integrated with the Teamwork/Ada graphical design tool and which supported code generation.

The FAADS software engineers are actually doing their development (i.e., the Ada design and Ada code generation) from within Teamwork/Ada. They are doing the design by creating Buhr diagrams and automatically generating the Ada skeleton code from the diagrams. They are using the DSE to create the detailed Ada code. The DSE does not allow code generated from the diagrams to be changed, without first changing the associated diagram and regenerating the Ada code. This assures that the code and the design documentation are always in synch.

The SDD documentation generation is being done with DocEXPRESS, a third party tool which is integrated within Teamwork. DocEXPRESS uses Teamwork's Documentation Production Interface (DPI) to generate 2167A-compliant documents.

Budgetary constraints prevented formal tool training. To partially offset this deficiency, the following measures were taken:

- Some team members attended the two week Paul Ward Real-time CASE curriculum. Although this class concentrated on method, it did utilize the diagrammatic conventions supported by Teamwork.
- The SEE Consultant (SC) provided a two-day informal hands-on training class, which included an introduction to the Teamwork environment.
- Team members referred to the Teamwork users manuals, including the limited amount of tutorial material.
- The SC circulated among the team and provided case-by-case suggestions.
- On two occasions, Cadre made one-day site visits, which included question and answer sessions and hands-on demonstration sessions.

ObjectMaker (Adagen)

Originally, IBM intended to use Adagen for software design. The reasons for shifting to Teamwork are discussed above.

Since FAADS future phases involved reuse of existing Ada code modules, some experimentation was conducted to evaluate Adagen's reverse engineering capability. The diagrams produced were of limited value by themselves, and since Teamwork had been chosen to support design, no attempt was made to modify them to make them usable. Future experimentation is planned with both Adagen's and Teamwork/Ada's reverse engineering capabilities.

DocEXPRESS

The FAADS documentation strategy called for producing documentation from the Teamwork design model, with minimal additional text publication work. DocEXPRESS supports this strategy by smoothing the interface between the Teamwork Document Production Interface (DPI) and the chosen text publishing software (Interleaf or FrameMaker).

The SC attended a three-day course offered by ATA (the vendor for DocEXPRESS) on 2167A software analysis and design. This course included an introduction to using DocEXPRESS, and was sufficient for the SC to set up tailored support for FAADS. Users required very little understanding of DocEXPRESS, since it was designed to be relatively transparent.

Interleaf

FAADS users had very little need to work within Interleaf, since most of the work was done by Teamwork and DocEXPRESS. None of the team received any relevant training.

The SDD is the only document produced with Interleaf (other project deliverables have been produced using Bookmaster, Word for Windows, and other tools). Interleaf was chosen for the SDD because it was one of two publication systems supported by Teamwork/DPI.

PVCS

PVCS was chosen to support configuration management aspects of the project. PVCS includes support for version management and configuration building. To date, only the latter capability is in use. None of the team received any PVCS training.

FAADS Lessons Learned from Introducing CASE Tools

The lessons learned by the FAADS project from using the IBM STARS Alpha SEE during the S-Increment are described below:

- Significant frontend budget allocation required for training and tools procurement and for installation and maintenance of SEE tools and network.

  Introduction of a new operating system (AIX), tools, and methods required considerable training costs and a significant learning curve for the project team, which are often underestimated in the original budget allocations.

- Strong management commitment and vision essential.

  Management must provide leadership and vision, as new (and often immature) SEE tools and methods are introduced into a project, to ensure that any initial negative (often valid) reactions are overcome and the necessary adaptations are made.

- New project roles required, e.g., system administrator for LAN and AIX, toolsmith for customizing and supporting use of tools.

  This is an especially difficult problem for a small site with limited access to support personnel.

- Single, integrated desktop access to SEE tools important for productivity and use of existing assets.

  A single virtual desktop access to all heterogeneous software/platforms preserves access to familiar tools, while taking advantage of existing site assets (rather than buying additional workstations or X terminals).

- Immaturity of methods (e.g., Buhr notation) and tools in design of large scale systems.

  Many of the notations currently being used for Ada design are relatively immature and evolving. As a result, problems occur when trying to scale up these notations to large systems, e.g., Ada design diagrams become very complex and difficult to understand.

- Immaturity of methods and tools in reverse engineering and reuse.

  Because of its immaturity, reverse engineering is an art, requiring careful tailoring of directives and manual post-processing. In addition, very little training is available on reverse engineering tools.

FAADS Lessons Learned from Using CASE Tools

The lessons learned on the FAADS project from using the IBM STARS Alpha SEE tools/methods are described below:

Teamwork Lessons Learned

- Some Teamwork tools/methods training would have been beneficial.

  Although the training described in the prior section was useful, users failed to pick up numerous time-saving techniques that would have been covered in Cadre's tool training classes. Although it is difficult to quantify, the training might have ended up paying for itself in the long run.

  Even more important than tools training, however, is methods training. Only a few team members received any methods training, and even that training concentrated on the analysis phase -- which was wasted to some extent since IBM ended up not being responsible for the SRS.

  The best possible course would a hybrid method/tool course, in which a specific method is taught using the tool as a hands-on vehicle. A one week Ada Design method course coupled with Teamwork/Ada would have been ideal.

- The value of prior experience.

  As echoed throughout the industry, there is no substitute for prior experience. This holds true with respect to methods, tools, and environments.

  Based on the reports of other projects, IBM had a basic understanding of the impact that would result from the number of changes being introduced for FAADS. These changes included:

  - Migration from a centralized, Vax-based environment to a networked AIX-based environment;
  - Use of several significant new tools;
  - Adaptation of the existing software methods and process to the above.

  The project felt that it would break even, at best, during the incorporation of these changes, but that the investment during the first phase of the contract (FAADS Model I) would result in higher quality for the Model I (which it has) and higher productivity during the subsequent phase (Model II). In retrospect, the impact was underestimated: the number of changes may have been too ambitious for a relatively small project. Unfortunately, it is impossible to quantify the productivity impact, since there have been too many other variables.

  Although the project is not yet complete, it does seem likely that the next phase will realize the improved productivity that was assumed when it was bid. This is due to several factors:

  - Most of the learning curve is over, and the team is acclimated to the adapted software process using the new environment and tools;
  - Many of the problems encountered during the first phase have either been solved or workarounds have been devised;
  - A Teamwork reuse base has been established for the next phase, providing a head-start in developing the three Model II CSCIs;
  - The performance of the Ada compilation system is so much better on the new RISC System/6000 than it was on the previous Vax system that much less time is lost waiting for compilations, simulation runs, and builds.

- Immaturity of Teamwork/Ada and DSE impacted their use.

  Teamwork/Ada, and particularly the DSE, are very new products. In addition, because of their enhanced functionality, the FAADS project opted to introduce these tools during their beta test phases. Both of these facts resulted in significant impacts due to bugs and problems in the use of the tools (e.g., crashes and loss of data).

- Availability of needed resources on configured hardware.

The FAADS project encountered some stability problems because the RISC System/6000 hardware was configured with marginal memory and disk space. Orders placed to remedy this problem could not be filled until late in the program because of high demand for RISC System/6000 hardware. For most of the program, Teamwork was running on machines with only 16MB memory; experience has shown that a minimum of 64 MB is needed for a Teamwork server machine, and that a minimum of 24 MB is needed for a user on a remotely connected RISC System/6000. Currently, the project is using a RISC System/6000 Model 550 for the server machine; this comfortably supports our average of three to four user sessions.

The architecture of the Teamwork product calls for a single model database on a host machine. There are then two methods of using Teamwork from another machine on the network: mounting the Teamwork directories using NFS (and running Teamwork locally), and remotely logging on to the server machine (and using the local machine as an X Server). Using the Model 550 as a host, users have found that the latter method is the most stable; and they pay no response time penalty to use it.

- Teamwork/Ada functionality

  The Teamwork/Ada graphical editor is responsive and robust. The Ada Structure Graph (ASG) notation implemented by the editor (based on the Buhr notation), however, while good at reflecting Ada code structure, has proven insufficient by itself to communicate the software design, and the customer was unhappy with it. The problems were that its notation was unfamiliar and that it does not adequately reflect the following aspects:

  - Data flow

    The notation shows packaging and invocation well, but falls short in showing accesses to data structures, and the functional interfaces among high-level software components.

  - Operational flow

    Using standard ASGs, there is no good way to show how the software components combine to respond to external stimuli (such as an operator action, or the arrival of a signal event).

  To supplement the design documentation, the team added high-level data flows (using Teamwork/SA), and a set of hybrid operational flow diagrams (using Teamwork/Ada).

- Teamwork/DSE functionality

  The users found that the DSE provides significant value added when compared with an ordinary text editor, since it understands Ada syntax. Among other benefits, it automates much of the formatting (reducing keystrokes), identifies syntax errors during text entry, and enhances on-screen readability of the code.

  Teamwork/DSE is integrated with the Teamwork/Ada model and enforces adherence to the design diagrams. Since the SDD documentation is also driven from the same model, this paradigm assures a level of agreement among the documentation, the design model, and the code. Most importantly, using Teamwork/DSE for Ada code development provides the capability to generate both the design documentation and the code from the same design (model) database, helping to assure agreement between them. This is a major strength of using Teamwork/Ada and DSE.

  While problems with the DSE product (bugs and limitations) have hampered full realization of the benefits of the approach, the team generally regards its use as a significant improvement to the Ada development process.

  As of this writing, Cadre has initiated an effort to improve the product. The major problems are:

  - The editor is prone to crashing, causing some loss of data. Users have adopted a practice of saving frequently.
  - Transitioning from the ASG editor to the DSE editor sometimes results in truncation of the Ada source code.

- The "pretty-print" rules adopted by the editor sometimes renders the code much less readable: users want the ability to selectively inhibit the reformatting, or to have more ability to modify the formatting rules.

- Teamwork/DPI Functionality

  Teamwork's Document Production Interface (DPI) is intended to make it possible to generate a document from the Teamwork model database. Most projects suffer from the "multiple, inconsistent databases" problem, and one of Teamwork's central appeals is the advertised ability to produce both documentation and code from a single database. The FAADS team has realized this goal to some extent, but several major problems remain. Some of these problems are addressed by the new Teamwork/DocGEN module, and the team plans to take advantage of its new features when producing the final version of the SDD.

  The following lists the major DPI problem areas:

  - Text Formatting

    DPI offers little capability to affect the formatting of the published text. Ideally, one would like the ability to enter text using the publication software (Interleaf in this case), since this would allow direct entry of lists, tables, etc., and since it would allow complete control over readability techniques such as italics and underscoring. Instead, the user must use Teamwork's rudimentary text editor and resort to a very limited set of DPI formatting commands.

  - Hierarchical Descent of the Software Structure

    DPI provides a powerful "parse_model" command that allows most of the SDD to be constructed automatically from the model. Parse_model starts from a specified node in the software component hierarchy (in this case a specific Ada Structure Graph diagram) and descends the subtree below this point -- embedding text and other pictures that are attached as notes to the ASG diagrams. The user controls the manner in which these notes are embedded using a format specification file. Unfortunately, there are several notable limitations:

    - ▲ At each level of the hierarchy, the diagrams are introduced in alphabetical order, according to the ASG diagram names; this is almost never the best order in which to introduce them from an understandability standpoint.

    - ▲ Once a parse_model descent is started, it continues to the bottom-most points in the hierarchy. This fact makes it very difficult to adhere to the 2167A DID for the SDD, which calls for a top-level depiction of the software in Section 3 and an intermediate- to low-level depiction in Section 4. This would seem to dictate two different design models, one for the high-level design and one for low-level design. The team decided to stick with the one-database philosophy, but this decision carried with it the need to develop some workarounds to the DPI limitations.

      Unfortunately, the decision also affected the CSC/CSU design structure. In Section 3, the SDD should decompose the software down to the CSC level, but DPI provides no way to cut the parse_model descent short. As a result, Section 3 descends down to the CSU level.

    - ▲ There is no provision for parsing the model multiple times in a single document with different formatting rules. The team was able to develop a work-around for this limitation, but it should be a part of the product.

    Unfortunately, none of the parse_model limitations are addressed in the new DocGEN product component.

  As of this writing the new DocGEN facility is available, and the team plans to take advantage of several of its features to improve the final version of the document:

  - New formatting features are now available, including the ability to construct lists and tables.

- It is now possible to compare earlier versions of documents with newer ones and generate a new document with change bars.

- Model Configuration Management (MCM) functionality

  Teamwork includes some basic CM features, retention of the past 16 versions of each model object, multi-user checkin/checkout of model objects, and the ability to baseline models (and to construct "derivative" models where modifications are separately maintained). These features have proved useful as far as they go: users have been able to work effectively as a team without worrying about losing data due to conflicts.

  Some improvements are needed, however, to fully enable the "single database" strategy (production of documents and code from a single Teamwork model). A standard software CM practice is to control software products in a hierarchical library structure, with higher levels containing previously released software and lower levels containing incremental changes awaiting testing so that they can move to higher levels of control. With this strategy, a method is needed to "promote" components from lower library levels into the higher ones. With conventional files, this can be accomplished by file moves from one directory to another (with suitable controls, of course). Similar functionality is provided with commercial CM tools such as PVCS and CCC.

  Ideally, the project should be able to manage the Teamwork model in the same way so that the Ada code levels can be kept in perfect sync with the corresponding model levels. Unfortunately, MCM offers no way to implement the "promote" function.

  Cadre has been receptive to this problem and is considering ways to address it. Another vendor, Softool, is currently working to tailor its product to Teamwork to provide this functionality in its commercial CM product, CCC.

ObjectMaker (Adagen) Lessons Learned

As mentioned earlier, the project originally intended to use Adagen to support the design process. As Cadre added Teamwork/Ada, and later DSE, the strategy changed to use Teamwork instead of Adagen. As a result, FAADS gained little experience in using Adagen, other than some initial experimentation with its capability to reverse engineer Ada code. It should be noted that Adagen has gone through significant improvements since FAADS early use of it.

DocEXPRESS Lessons Learned

- DocEXPRESS Functionality

  DocEXPRESS simplifies the transition between Teamwork/DPI and publication software (Interleaf or FrameMaker) -- with specific support for producing 2167A documentation. It provides:

  - Additional Teamwork menus for building documents (by itself, DPI must be invoked outside of Teamwork);
  - Predefined templates including boilerplate, to give a headstart for the standard 2167A documents;
  - Consistent formatting among documents, conforming to DID standards as to headings, section and page numbering, etc.

  Unfortunately, DocEXPRESS executes on top of Teamwork/DPI and inherits underlying limitations of that tool. It should be noted that substantial improvements in both DocEXPRESS and Teamwork document generation have been made since the time of this experience. Even with these improvements, however, generation of deliverable-quality documentation remains a significant challenge.

- Doc EXPRESS Documentation and Support

  The users manual is very clear and provides specific, detailed usage instructions.

One of the biggest advantages of using the product is the excellent support offered by the vendor, ATA. ATA personnel have a significant experience base in systems, and specifically in 2167 and 2167A systems. They are also intimately familiar with both Teamwork and the text publication software.

Armed with this experience, they were very responsive in providing product support, including advice on methodology.

Interleaf Lessons Learned

- Publishability of Documents

  In accordance with the "single database" strategy, the team spent minimal time using Interleaf. As discussed elsewhere, however, due to limitations of DPI, the resulting SDD was marginally publishable. It would have been possible to greatly enhance the appearance of the document by using the considerable power of Interleaf, but to do so would have meant introducing multiple databases.

- Tool Integration

  Ideally, tools for modeling (e.g., Teamwork) and documenting (e.g., Interleaf) should be more closely integrated. Several vendors are pursuing this concept. One possible approach would be to allow the Teamwork user to use the publication software while entering text. Another approach would be to give the user transparent navigation between the publication software and the modeling tool.

- Licensing

  There are two means of licensing Interleaf: networked and node-locked. With the networked method, Interleaf can run anywhere on the network, but only a maximum number of users can run it at one time. With the node-locked method, Interleaf only runs on one machine, but any number of users can use it at one time. The cost of the node-locked license is the same as a one-user networked license.

  Because Interleaf is only used for the SDD, and because one user is assigned the task of building the document, it is rare that more than one user is accessing Interleaf at once. Hence, the node-locked method has proven more economical for FAADS.

  Interleaf is installed on the server machine. Users on remote nodes can connect to the product via remote logon, using their local machines as X Servers. Given the limited need for interactive use, response time using this approach has proven adequate.

PVCS Lessons Learned

- Version Manager

  As of this report, the project has not yet incorporated the Version Manager. The site has an established practice of using a multi-level library scheme for controlling incremental software releases. With this scheme, all files are separately maintained, and promotions to higher library levels are accomplished by moving integral files.

  In contrast, the PVCS Version Manager uses monolithic files to store multiple versions of each software component. This method represents a significant departure for the organization, and its incorporation is still under study. Advantages of using the Version Manager would include:

  - More concise storage of multiple versions of source files by using "delta" techniques; and
  - Opportunity to store all archived versions on-line (because of the reduced storage requirements) and to readily reconstruct past archived configurations.

- Configuration Builder

The PVCS Configuration Builder is based on the Unix make, with additional enhancements. PVCS has enabled straightforward automation of the software compilation and build process, including the handling of multi-level libraries.

If the project elects to move to the Version Manager (see above), PVCS includes provisions for integrating the Configuration Builder capabilities with minimal effort.

- Integration

  PVCS by itself provides no capability for integrating Teamwork model artifacts with the code artifacts (the need for doing this has been discussed earlier, in the topic of Teamwork's Model Configuration Management facilities).

  Another vendor, Softool, is developing integrated support for Teamwork and code for its CM tool, CCC.

- Multi-Platform Considerations

  One of the FAADS software CSCs is being implemented in the C language, and it is being developed on an IBM PS/2 since there is no appropriate RISC System/6000 compiler for the processor on which the CSC executes. Although it is desirable to have a single CM database for the entire project, this separation of platforms poses problems.

  At present, the PS/2 software construction process is segregated from the RISC System/6000 process for Ada code. The project is assessing the possibility of integrating the two processes, using NFS on the PS/2 to mount the RISC System/6000 code libraries.

  Should the project also adopt the PVCS Version Manager, Intersolv markets a networked version of PVCS for the PS/2 which would allow common usage of the PVCS database from both platform types.

Miscellaneous Lessons Learned

- The impact of introducing multiple changes/technologies was underestimated (the number of changes may have been too ambitious for a relatively small project). These changes included:
  - Migration from a centralized, Vax-based environment to a networked AIX-based environment.
  - Use of several significant new tools, e.g., Ada, Teamwork, Interleaf.
  - Adaptation of the existing software methods and process to the above.
- One of the greatest performance improvements accrued from upgrading the SEE CPU "horsepower":
  - Ada compilations
  - System builds
  - Test runs using the target computer emulator
- Higher quality was realized in the first phase of the contract (FAADS Model I) and higher productivity is expected for subsequent phase (Model II), due to:
  - Most of the learning curve is over, and the team is acclimated to the adapted software process using the new environment and tools
  - Many of the problems encountered during the first phase have either been solved or workarounds have been devised
  - A Teamwork reuse base has been established for the next phase, providing a head-start in developing the three Model II CSCIs

Summary of Combined Lessons Learned on STARS Alpha Test Projects

In this concluding section, we reflect on all three projects and attempt to distill some of the common lessons learned.

Impediments to Change/Remedial Strategies

All of the projects were groundbreakers, and (to push the metaphor a bit) they all encountered boulders as they were getting underway. The following impediments are representative of the type that may be encountered on any similar project attempting to inject new SEE approaches. For each category, we include some constructive ideas on how a project might attempt to prepare for and offset these impediments.

- Problem: Inertia
  - People are comfortable with the existing process
  - There is a tendency to subvert new methods to old ways of thinking
  - Attitude is all-important

  Strategies:
  - Insure strong support, vision from management and tech leads
  - Enlist early support, involvement from customer
  - Involve people in planning, preparations
  - Develop phased implementation plan, tailored to group
  - Consider formal Technology Transition training
- Problem: Overblown Expectations
  - Marketing hype, overzealous advocates are common
  - Unrealistic hopes lead to disillusionment
  - Unanticipated costs can blossom

  Strategies:
  - Interview teams with real-project experience
  - Try out SEE, tools, methods, process on pilot project
  - Carefully weigh degree of change against cost uncertainties
  - Explicitly plan for each cost category (see checklists, below)
  - Expect no productivity increase on first system

- Avoid "panacea mentality":
  - ▲ New methods and tools are no substitute for domain expertise
  - ▲ Poorly thought-out SEE strategy can degrade effectiveness
- Key: Match Resources to Ambitions
  - Small projects should focus on incremental change
  - Large projects can handle more change, but only with careful planning

Combined Lessons Learned: Planning

Because of the potential risks in introducing new approaches, and because it's vital to get off to as sound a start as possible, planning is especially important. Here are some of the considerations to take into account when planning the project.

- Anticipate Essential Startup Activities
  - Clear identification of methods
  - Methods training
  - Tool evaluation, selection
  - Tool adaptation/integration design & implementation
  - Tool training
- Assess Cost of SEE Realistically, including:
  - H/W components and networking
    - ▲ Consider wiring, installation, checkout, support, maintenance
    - ▲ Insure necessary computing resources & seats to deliver tool capability to users with adequate response time
    - ▲ If possible, install trial configuration before final planning
  - S/W
    - ▲ Be sure number of licenses will support planned roles
    - ▲ Don't forget software maintenance (typically 15%/yr)
  - Adaptation and Integration Expenses

    Note: This can turn out to be extensive:
    - ▲ Each tool typically requires tailoring to fit process
    - ▲ Varying degrees of integration required for tools to work together
    - ▲ Databases typically stretch across heterogeneous platforms
    - ▲ Requirements will continue to emerge throughout project
  - Administration: H/W, Networking, and Tools
  - Plan for Ongoing Roles, such as:
    - ▲ SEE and tool administrators
    - ▲ Adaptation & integration evolution and support

- ▲ Methodology consultants (motivated, knowledgeable, proactive)

- Involve the Customer Early
  - ▲ Agree on approach and required mutual investment
  - ▲ Consider including customers in training sessions
  - ▲ Tailor documentation/deliverable plan (e.g., 2167A)
- Plan for Balanced Learning Approach
  - ▲ Classes (methods, tools, SEE)

    (Note: "Just-in-time" Training is most effective)
  - ▲ Domain-specific workshops, with expert consulting
  - ▲ Pilot project
  - ▲ Hands-on Experience

Combined Lessons Learned: Maintain a Healthy Respect for Murphy's Law

When introducing a significant amount of new technology into a project, it is definitely not the time to utter the phrase "...now, if everything goes well...". The more opportunities for the unexpected to arise, the more opportunities for things to go wrong. Each of the three Alpha projects found ample proof of this principle, and the following examples are cited to provide a flavor.

- Be Wary of Beta Test Versions and Initial Releases

  When you find out the limitations of a tool you've decided to use, you may be anxious to get the next version, perhaps even a Beta (or even an Alpha?). Bear in mind, however, that the inevitable bugs will compound the group's problems of assimilation. If things are bad enough, it might kill the initiative before it gets a real start. Before yielding to the temptation to get the latest and greatest, consider the vendor's past quality record.

- "You only know what you're in for when you're in it"

  This point cannot be overemphasized. Students of calculus often find out the hard way that you don't really learn the subject until you do the problems, and the same principle applies to learning to apply new SEE approaches. All three of the projects rediscovered this as they found that what seemed so smooth in the visionary's pitch (or the vendor's sales literature) had lots of bumps in practice. Experiences from FAAD/NCTR1 are offered as illustrations. These are given without elaboration; interested readers are invited to contact the authors for specifics.

  - Bugs & Kinks

    (e.g., Transition from Teamwork[1] graphical editor to DSE editor sometimes resulted in truncated Ada code files)
  - Gaps

    (e.g., Buhr notation not sufficient for representing the design. This point is discussed in more depth in the next subsection.)
  - Misapprehension of function

---

[1] Please note: these problems are not meant as criticisms of Teamwork; rather, they are meant to illustrate the problems that can be expected from any set of tools.

(e.g., Built-in Teamwork CM did not mesh with the organization's hierarchical library process)

- Integration with other parts of the SEE

  (e.g., Project legacy documents in Bookmaster, new documents in Interleaf)

Combined Lessons Learned: Technical Tidbits

This subsection lists some of the more interesting and salient technical points that might turn out to be of practical value to new projects.

- Key Objective: Retain Currency of Design as Code Evolves
  - GPS/ACE strategy: PDL maintained in Rational code; diagrams produced from code via reverse engineering
  - FAADS strategy: Teamwork/Ada & DSE: models & code kept in lockstep
- Key Objective: Provide User with Single Desktop Access to SEE Assets
  - GPS, ACE used RISC System/6000 workstations and PS/2s
  - FAADS supplemented workstations with PS/2 with MS-Windows X Server
- Methods Lessons Learned
  - Buhr notation (as per Teamwork/Ada) not sufficient for design
    - ▲ Describes static Ada structure
    - ▲ Additional diagrams needed for
      - △ Interfaces and dynamic behavior
      - △ Operational flows in response to usage scenarios
  - Reverse Engineering Requires Manual Assistance
  - Making diagrams readable
  - Discovering and reflecting interfaces and dynamic behavior
- Documentation Consistently Proved Harder to Produce than Expected

Combined Lessons Learned: Potential Rewards

This section has thus far emphasized caution, and the reader may have begun to conclude that the authors are against the introduction of change. On the contrary, we believe that despite growing pains, the Alpha Projects have shown that the future of automated support of the software process is very promising.

To help make this point, this final subsection is devoted to one of the key positive lessons learned: that there are substantial potential rewards for an organization that manages to weave its way through the obstacle course without crashing.

- Significant Morale Boost
  - Upgraded technology = = > upgraded skills
  - Willingness of management to invest in improving workplace
- More Effective New Process

- Better team communication/coordination
- Higher individual and team productivity
- Better quality work products

18th Annual Software Enginnering Workshop
Lessons Learned Applying CASE Methods/Tools To Ada Software Development Projects

December 1, 1993

Maurice H. Blumberg
(301)240-6018
blumberm@wmavm7.vnet.ibm.com
Dr. Richard L. Randall
(719)554-6597
randallr@wmavm7.vnet.ibm.com

STARS Project
IBM Federal Systems Company
800 N. Frederick Ave.
Gaithersburg, Md. 20879

Outline of Talk

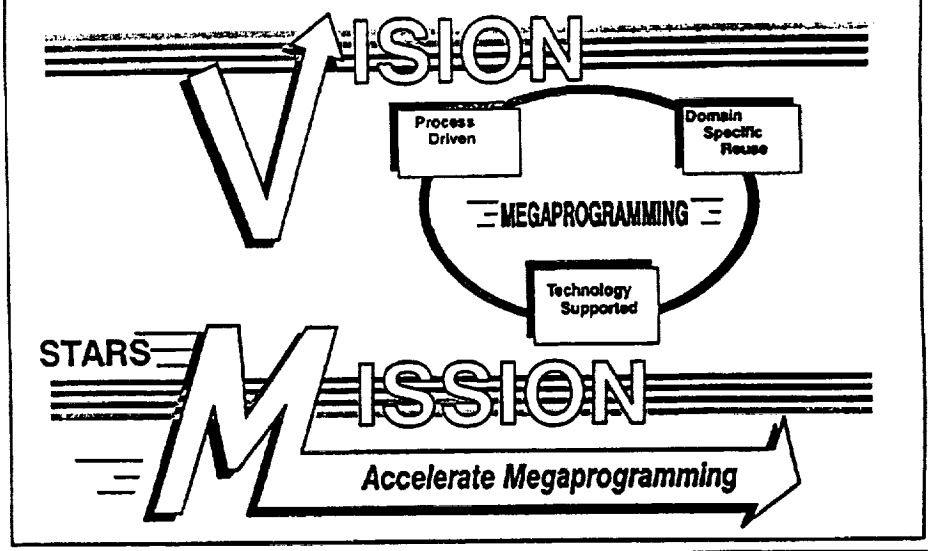
- Overall Context Setting - STARS Program
  - STARS Vision/Mission
  - STARS Strategy
- Lessons Learned Context - Alpha Test Projects Selected
  - GPS
  - ACE
  - FAADS
- Summary of Combined Lessons Learned from Alpha Test Projects
- Project-by-Project Lessons Learned

IBM December 1, 1993

STARS Program Overview

MEGAPROGRAMMING – AN EMERGING PARADIGM

DARPA



IBM December 1, 1993

STARS Program Overview

MEGAPROGRAMMING SUPPORT ELEMENTS

DARPA

Process Driven

- Guided by a defined process.
  - Developed from reusable process building blocks.
  - Adaptable to meet project/product goals.
  - Promotes collaboration and team work.
- Supported by tools.
- Supports continuous improvement in process and product.

Domain-Specific Reuse

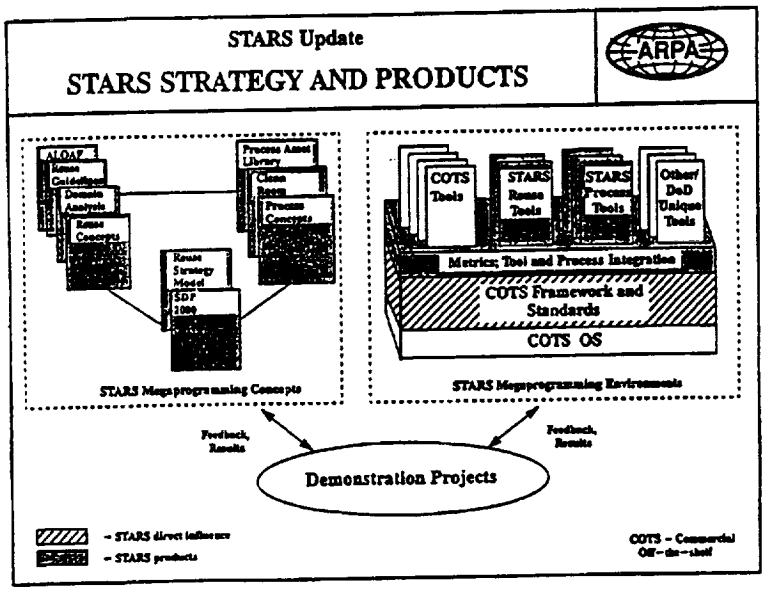
- Guided by reuse process.
- Based on application domain architecture.
- Systems composed from reusable assets.
- Assets include any/all life-cycle artifacts.
- Supports continuous improvements in reuse process/products.

MEGAPROGRAMMING

Technology Supported

Process/Reuse Packaged in a SEE

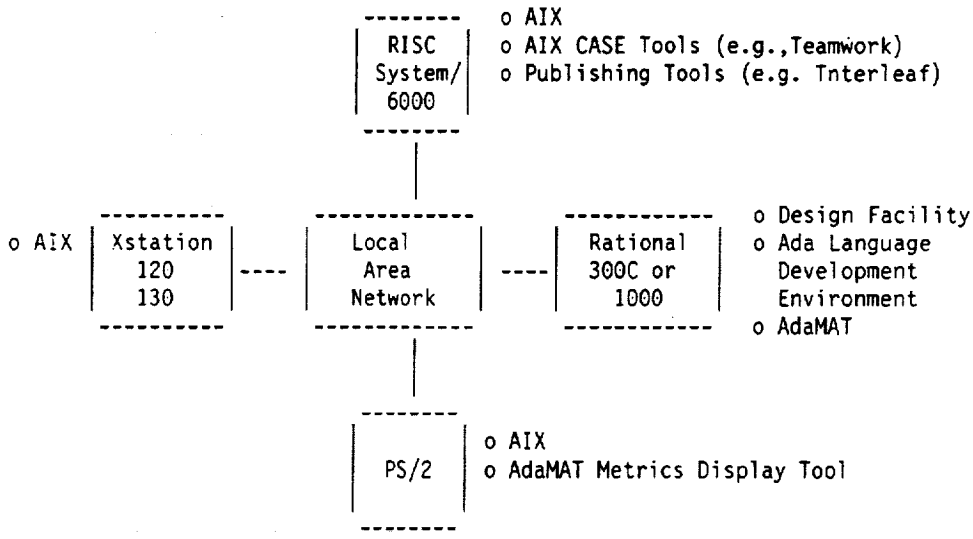
- Based upon open architecture framework.
- Adaptable approach for incorporating new technologies.
- Continuous improvement in portability, adaptability, reliability, and scalability.



Purpose of Alpha Test Projects

- Gain early experience and feedback in the use of the IBM STARS Alpha SEE Solutions
- Provide vehicle for early technology transfer of IBM STARS capabilities
- Be a precursor for STARS Demonstration Projects in defining:
  - A technology transfer process
  - How to support projects using a SEE
  - How to capture lessons learned information

IBM STARS Alpha SEE Solution



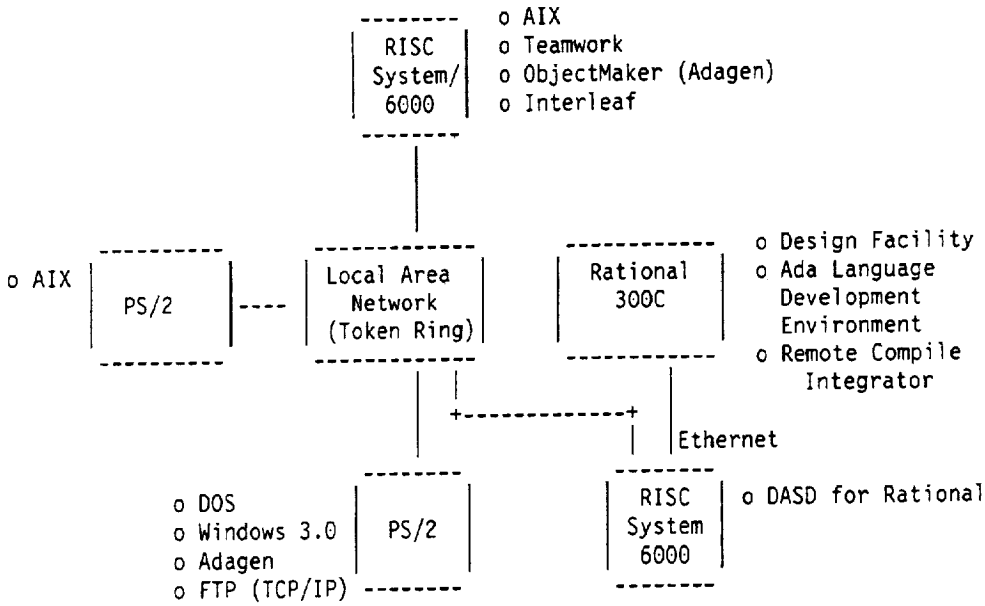
Current Alpha Test Projects

- Global Positioning System (GPS)
- Ada CASE Engineering (ACE)
- Forward Area Air Defense (FAAD) Electronic Support Measure (ESM) Non-cooperative Target Recognition (NCTR) System (FAADS)

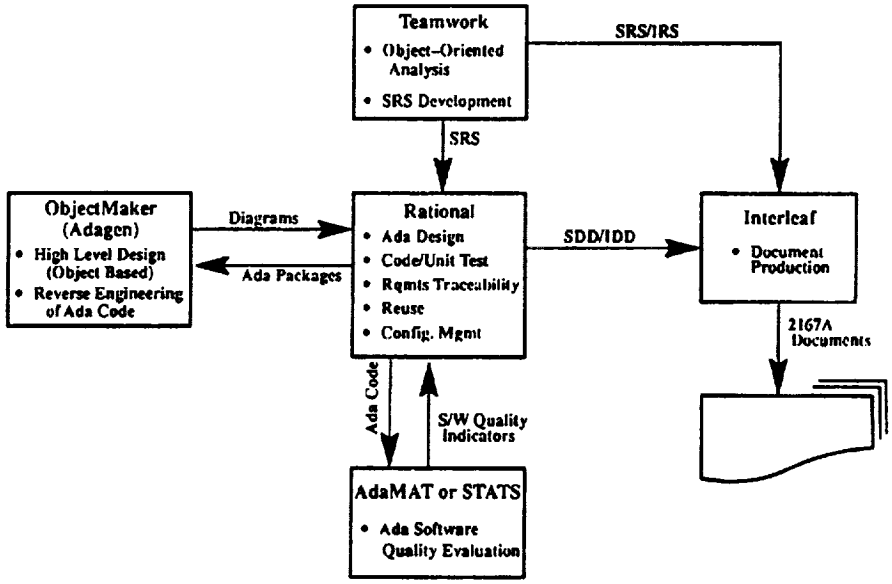
Alpha Test Projects SEE Tool Usage

Life Cycle Activity	GPS	ACE	FAADS
Analysis	Teamwork	Teamwork STATEMATE	N/A
Design	Adagen Rational	Adagen Rational	Teamwork/ Ada & DSE
Implementation	Rational	Rational	TLD Ada Compiler
Document Generation	Teamwork DocEXPRESS Interleaf Rat.	Rational	Teamwork DocEXPRESS Interleaf
Reverse Engineering	Adagen	Adagen	Adagen
Requirements Traceability	Rational	Rqt RTrace RTM Rat.	Manual

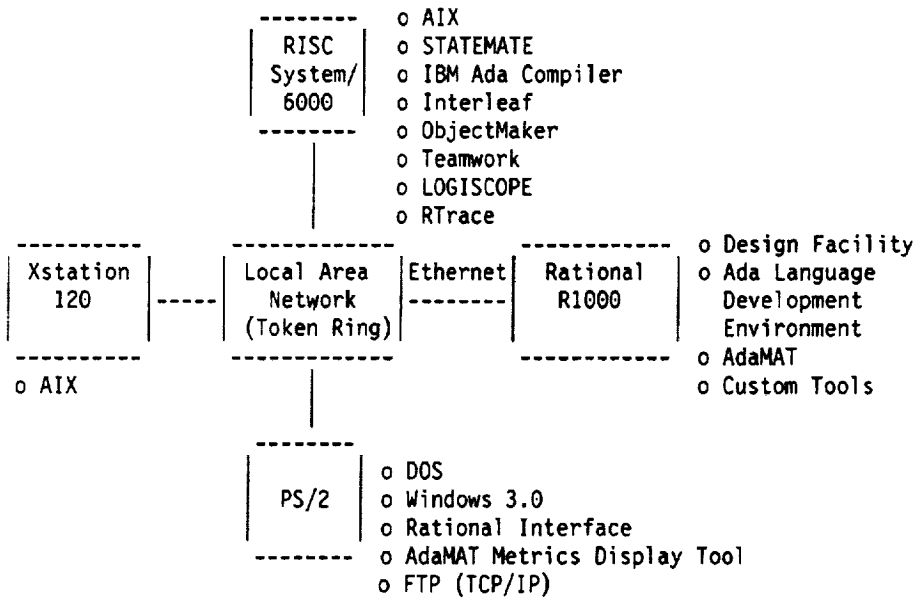
GPS Hardware/Software Configuration



GPS Ada Design / Development Environment



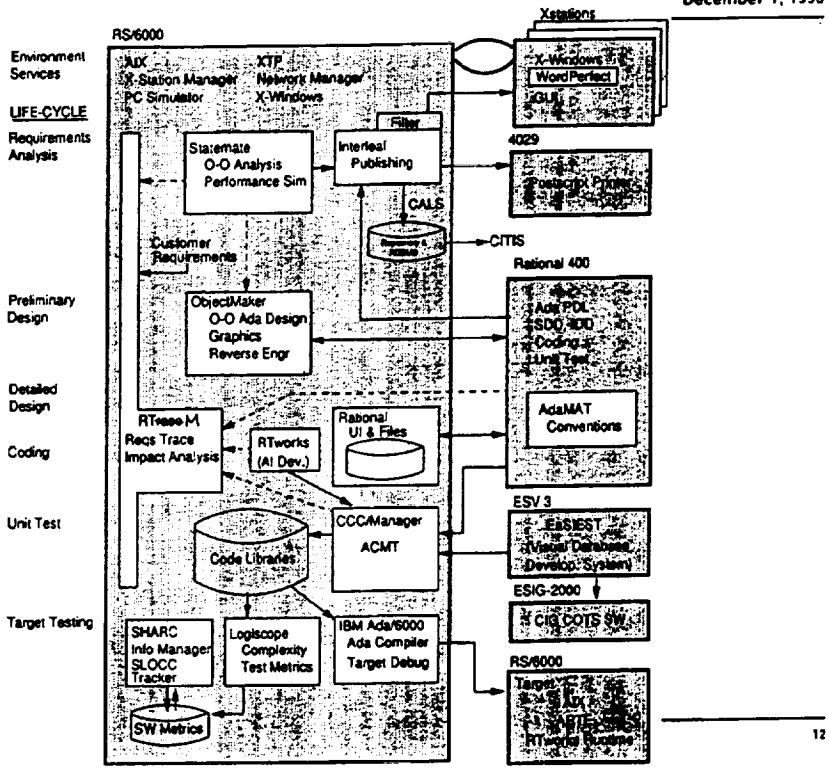
ACE SEE



IBM

CCTT SOFTWARE DEVELOPMENT ENVIRONMENT

December 1, 1993

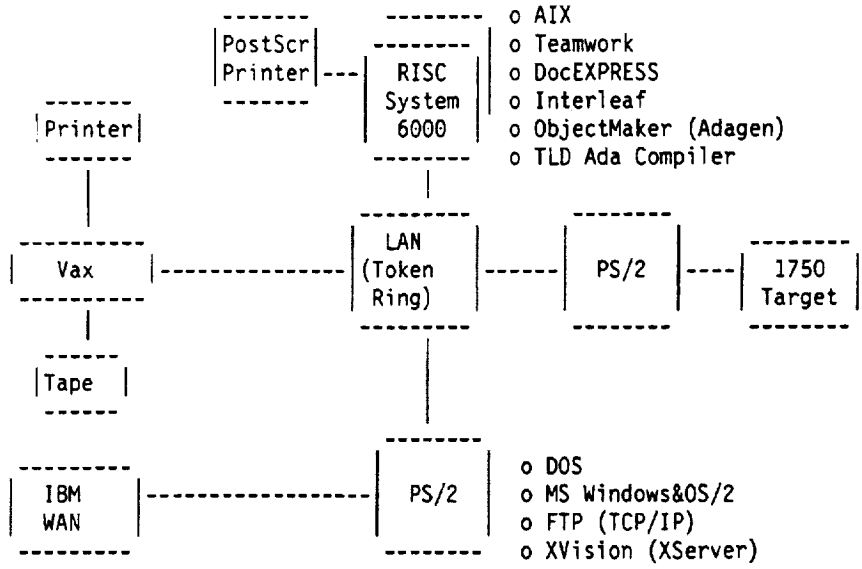


12

IBM

December 1, 1993

FAADS SEE



Lessons Learned Applying CASE Methods/Tools To Ada Software Development Projects 13

Summary of Combined Lessons Learned on Alpha Test Projects

Impediments to Change/Remedial Strategies

o Problem: Inertia
- People are comfortable with the existing process
- There is a tendency to subvert new methods to old ways of thinking
- Attitude is all-important

Strategies:
- Insure strong support, vision from management and tech leads
- Enlist early support, involvement from customer
- Involve people in planning, preparations
- Develop phased implementation plan, tailored to group
- Consider formal Tech Transition training

o Problem: Overblown Expectations
- Marketing hype, overzealous advocates are common
- Unrealistic hopes lead to disillusionment
- Unanticipated costs can blossom

Strategies:
- Interview teams with real-project experience
- Try out SEE, tools, methods, process on pilot project
- Carefully weigh degree of change against cost uncertainties
- Explicitly plan for each cost category (see checklists, below)
- Expect no productivity increase on first system
- Avoid "panacea mentality":
  -- New methods and tools are no substitute for domain expertise
  -- Poorly thought-out SEE strategy can degrade effectiveness

o Key: Match resources to ambitions
- Small projects should focus on incremental change
- Large projects can handle more change, but only with careful planning

Combined Lessons Learned: Planning

- o Significant Startup required, including
  - Clear identification of methods
  - Methods training
  - Tool evaluation, selection
  - Tool adaptation/integration design & implementation
  - Tool training

- o Assess cost of SEE realistically
  - H/W components and networking
    - -- Consider wiring, installation, checkout, support, maintenance
    - -- Insure necessary computing resources & seats to deliver tool capability to users with adequate response time
    - -- If possible, install trial configuration before final planning
  - S/W
    - -- Be sure number of licenses will support planned roles
    - -- Don't forget software maintenance (typically 15%/yr)

- Adaptation and integration expenses - note: can be extensive
  - -- Each tool typically requires tailoring to fit process
  - -- Varying degrees of integration required for tools to work together
  - -- Databases typically stretch across heterogeneous platforms
  - -- Requirements will continue to emerge throughout project
- Administration: H/W, networking, and tools

- o Plan for ongoing roles, such as:
  - SEE and tool administrators
  - Adaptation & integration evolution and support
  - Methodology consultants (motivated, knowledgable, proactive)

- o Involve the Customer early
  - Agree on approach and required mutual investment
  - Consider including customers in training sessions
  - Tailor documentation/deliverable plan (e.g., 2167A)

- o Plan for balanced learning approach
  - Classes (methods, tools, SEE)
    (Note: "Just-in-time" Training is most effective)
  - Domain-specific workshops, with expert consulting
  - Pilot project

Combined Lessons Learned: Maintain a Healthy Respect for Murphy's Law

- Be wary of Beta test versions and initial releases
- "You only know what you're in for when you're in it"

  Examples from FAAD/NCTR1:
  - Bugs & Kinks
    (e.g., Transition from Teamwork* graphical editor to DSE editor sometimes resulted in truncated Ada code files)
  - Gaps
    (e.g., Buhr notation not sufficient for representing the design)
  - Misapprehension of function
    (e.g., Built-in Teamwork CM did not mesh with the organization's hierarchical library process)
  - Integration with other parts of the SEE
    (e.g., Project legacy documents in Bookmaster, new documents in Interleaf)

---

* Please note: these problems are not meant as criticisms of Teamwork; rather, they are meant to illustrate the problems that can be expected from any set of tools.

Combined Lessons Learned: Technical Tidbits

- Key objective: retain currency of design as code evolves
  - GPS/ACE strategy: PDL maintained in Rational code; diagrams produced from code via reverse engineering
  - FAADS strategy: Teamwork/Ada & DSE: models & code kept in lockstep
- Key objective: provide user with single desktop access to SEE assets
  - GPS, ACE used RISC System/6000 workstations and PS/2s
  - FAADS supplemented workstations with PS/2 with MS-Windows X Server
- Methods Lessons Learned
  - Buhr notation (as per Teamwork/Ada) not sufficient for design
    - -- Describes static Ada structure
    - -- Additional diagrams needed for
      - --- Interfaces and dynamic behavior
      - --- Operational flows in response to usage scenarios
  - Reverse engineering requires manual assistance
    - -- Making diagrams readable
    - -- Discovering and reflecting interfaces and dynamic behavior
- Documentation consistently proved harder to produce than expected

Combined Lessons Learned: Potential Rewards

o Significant morale boost possible
  - Upgraded technology ==> upgraded skills
  - Willingness of management to invest in improving workplace

o More effective new process
  - Better team communication/coordination
  - Higher individual and team productivity
  - Better quality work products

Project-by-Project Lessons Learned

Global Positioning System (GPS)

Description

- GPS project is in its 13th year of development and follow-on contracts.
- The current system consists of approximately 1 million SLOC, mostly in JOVIAL.
- Current development effort is for hardware and software to enhance GPS ground support system, including:
  - Development of Software Requirements Specifications (SRS)
  - Development of Software Design Documents (SDD) and associated Ada code
  - Development of future Computer Software Configuration Items (CSCIs) on RISC System/6000 work stations, using Ada (planned but canceled)

GPS Ada Lessons Learned from Introducing CASE Tools

- Significant start up preparation and cost for a new Ada project
- Customization of tools (e.g., Rational) requires significant resources
- Choose a project methodology and train developers early
- Have engineers use object oriented analysis for specifications
- Use Ada as the design language (design compiles)
- Preserve ability to extract PDL after code completion
- Agree with customer on diagramming and PDL techniques early
- Plan for a target code version and redoing unit tests on target
- Need additional personnel roles:
  - Rational System Administrator
  - Rational CMVC/RCF Administrator
  - Rational Design Facility Customizer
  - Adagen Support Expert

GPS Ada Lessons Learned from Using CASE Tools

Teamwork Lessons Learned

- An understanding of the basic capabilities of Teamwork was gained without formal classroom training (one person).
- Formal classroom training is required for understanding object-oriented method (Shlaer/Mellor) used by Teamwork.
- Generating an SRS which satisfies the customer's 2167A DIDs (Data Item Descriptions) requires considerable tailoring of the Teamwork templates.

DocEXPRESS Lessons Learned

- DocEXPRESS simplified generation of 2167A compliant documentation
- DocEXPRESS required considerable enhancements (by the DocEXPRESS vendor) to generate an SRS which satisfied the customer's DoD-STD-2167A DIDs (e.g., provide requirements traceability matrices).
- DocEXPRESS documentation and support are of high quality.

ObjectMaker (Adagen) Lessons Learned

- For its limited use on GPS (primarily conceptual design and reverse engineering), an understanding of the capabilities of Adagen was gained without formal classroom training.
- Reverse engineering of Ada code to create design diagrams for software design documents (SDD) ensured that the Ada graphical diagrams in the SDD were consistent with the Ada source code.
  - Education of the customer was required to gain their acceptance of Ada Structure Diagrams in the SDD.
  - Some manual editing of the reverse engineered diagrams generated by Adagen was required (to simplify and improve readability and satisfy the customer).

Rational Lessons Learned

- The Rational development environment was very effective in developing and testing of Ada code.
- Significant training is required to become proficient in the use of the Rational development environment.
- Expense and overhead of supporting the Rational development environment is high.
- A significant effort was required to customize the Rational Design Facility (RDF) to generate the GPS 2167A Software Design Documents.

Integration Lessons Learned

- Integrating tools to build an environment to support the entire life cycle is difficult.
- Generating documents automatically from CASE tools does not satisfy the requirement for page integrity.

Ada CASE Engineering (ACE)

Description

- Internal FSD project
- Setup CASE Tool Environment Laboratory in Manassas
- Performs ongoing evaluations of tools and methods that can improve Ada software development (including maintenance)
- Provides education for tools and methods
- Supports new Ada projects that use CASE tools and Rational Ada development environment
  - Fixed Distributed System (FDS)
  - Advanced Training System (ATS)
  - Global Positioning System (GPS)

ACE Lessons Learned from Introducing CASE Tools

- The single most important key to the success of a project is still to understand the problem thoroughly.
- Adequate training in tools/methods must be provided.
- New methods and tools require considerable time to learn and this time must be allocated to a project schedule.
- Tools require considerable lead time before they are operational.
- New methods/tools need to have a strong project advocate.
- A project should have a 'toolsmith' who can customize tools to the project when necessary.
- Consider whether a tool/method might not 'scale up' to a large project.
- Having tools available in the office via networking is a productivity enhancer.
- New tools and methods should not be seen as a panacea.

ACE Lessons Learned from Using CASE Tools

STATEMATE Lessons Learned

- STATEMATE panel generator and Ada Prototyper provide very useful and informative modeling views (e.g., for specification execution (animation), network and processor performance, and user I/F prototyping).
- Language and semantics of STATEMATE require a steep learning curve.
- Definition of STATEMATE naming conventions is very important.

RTrace Lessons Learned

- RTrace supports 2167A requirements traceability.
- RTrace is easy to use and should not require formal education.
- RTRACE is a standalone tool and is not integrated with any CASE tool.
- Projects using RTRACE will need a "guru" to customize reports and perform tool administration functions.
- The current release of RTRACE does not support any automated configuration management or version control.

Miscellaneous Lessons Learned

- Many tools need to be customized before they can be used on a project. This will require a project "toolsmith".
- Most CASE tools have a significant learning curve. Adequate training and learning time are required before users will become proficient in using CASE tools/methods.
- Bringing users onto a technology transition oriented project or a pilot project, before their real use of the tool is required, eases the learning process and makes the user more receptive.
- A course in how to write a good specification improved the quality of specifications produced by the developers.

FAADS

Description

- Development of hardware and software for a passive ESM system to support tactical forward area defensive weapons platforms in detecting airborne threats and cueing weapons operators.
- Magnavox is the prime contractor.
- IBM is the software developer.
  - Software to be developed on RISC System/6000 workstations, but will run on 1750A processors.
  - Software to be developed in two phases with planned reuse of Ada code in the second phase.

FAADS Ada Lessons from Introducing CASE Tools

- Significant frontend budget allocation required for training and tools procurement and for installation and maintenance of SEE tools and network.
- Strong management commitment and vision essential.
- New project roles required, e.g., system administrator for LAN and AIX, toolsmith for customizing and supporting use of tools.
- Desktop access to SEE tools important for productivity and use of existing assets.
- Methods and automated support are still immature, e.g.,
  - Graphical design depictions for large Ada systems (Buhr diagrams are not sufficient).
  - Reverse engineering (significant amount of manual work needed to supplement automatically generated diagrams).

FAADS Lessons Learned from Using CASE Tools

Teamwork Lessons Learned

- Combined Teamwork tools/methods training would have been beneficial
- Immaturity of Teamwork/Ada and Teamwork/DSE impacted their use
- It is important to ensure that needed resources on configured hardware are available for adequate tool stability and performance
- Teamwork functionality had both strengths and weaknesses:
  - Teamwork/Ada
  - Teamwork/DSE
  - Teamwork/DPI
  - Teamwork/MCM

DocEXPRESS Lessons Learned

- DocEXPRESS simplified the transition between Teamwork/DPI and publication software (Interleaf) and provided specific support for producing DoD-STD-2167A compliant documentation.
- DocEXPRESS executes on top of Teamwork/DPI and inherits underlying limitations of that tool.*
- DocEXPRESS documentation and support are of high quality.

---

* Substantial improvements in both DocEXPRESS and Teamwork document generation have been made since the time of this experience. Even with these improvements, however, generation of deliverable-quality documentation remains a significant challenge.

Miscellaneous Lessons Learned

- The impact of introducing multiple changes/technologies was underestimated (the number of changes may have been too ambitious for a relatively small project). These changes included:
  - Migration from a centralized, Vax-based environment to a networked AIX-based environment.
  - Use of several significant new tools, e.g., Ada, Teamwork, Interleaf.
  - Adaptation of the existing software methods and process to the above.
- One of the greatest performance improvements accrued from upgrading the SEE CPU "horsepower":
  - Ada compilations
  - System builds
  - Test runs using the target computer emulator

- Higher quality was realized in the first phase of the contract (FAADS Model I) and higher productivity is expected for subsequent phase (Model II), due to:
  - Most of the learning curve is over, and the team is acclimated to the adapted software process using the new environment and tools
  - Many of the problems encountered during the first phase have either been solved or workarounds have been devised
  - A Teamwork reuse base has been established for the next phase, providing a head-start in developing the three Model II CSCIs