N93-11943

THE DEVELOPMENT AND TECHNOLOGY TRANSFER OF SOFTWARE ENGINEERING TECHNOLOGY AT JOHNSON SPACE CENTER

C. L. Pitman, D. M. Erb*, M. E. Izygon**, E. M. Fridge III, G. B. Roush, D. M. Braley, and R. T. Savely
NASA/Johnson Space Center
Software Technology Branch/PT4
Houston, TX 77058

ABSTRACT

The Unites States' big space projects of the next decades, such as Space Station and the Human Exploration Initiative, will need the development of many millions of lines of mission critical software. The Johnson Space Center (JSC) is identifying and developing some of the Computer-Aided Software Engineering (CASE) technology that NASA will need to build these future software systems. The goal of this research and development is to improve the quality and the productivity of large software development projects. This paper outlines new trends in CASE technology and describes how the Software Technology Branch (STB) at JSC is endeavoring to provide some of these CASE solutions for NASA. Key software technology components include knowledge-based systems, software reusability, user interface technology, reengineering environments, management systems for the software development process, software cost models, repository technology, and open, integrated CASE environment frameworks. The paper presents the status and long-term expectations for CASE products. The STB's Reengineering Application Project (REAP), Advanced Software Development Workstation (ASDW) project, and software development cost model (COSTMODL) project are then discussed. Some of the general difficulties of technology transfer are introduced, and a process developed by STB for CASE technology insertion is described. Finally, plans for future work are outlined.

1. INTRODUCTION

The Space Station Freedom Program and the Human Exploration Initiative will require the development, use, and maintenance of software systems containing millions of lines of code. These software systems present severe technical and managerial challenges. For example, the development and maintenance of these systems will require hundreds of well-trained software engineers, often working in parallel, and this will require well-managed, disciplined software development and maintenance processes. NASA is aware of these challenges and understands that new software engineering technology will be needed in the coming years. This paper describes ongoing work within the Software Technology Branch (STB), Johnson Space Center (JSC), to identify and develop some of the software technology required to meet these challenges and to transfer it to all of NASA.

The main goals of this work are to improve the productivity of software developers, users, and maintainers and to improve the quality of the software systems. This is to be accomplished through the enforcement of good software engineering principles and the use of Computer-Aided Software Engineering (CASE) throughout the whole software life cycle. Within an open, fully-integrated CASE environment, knowledge-based technology will be used to guide the entire software development process. Where necessary, large existing programs that have become too costly to maintain will be reengineered. Software and data reusability and application generators are also key elements of this approach to improve software quality and productivity.

2. SOFTWARE ENGINEERING TECHNOLOGY

One of the biggest thrusts in software engineering technology today is the push to develop an open, fully-integrated CASE environment -- i.e., a CASE environment wherein different vendors' hardware and software components can work together effectively. Many of the world's largest computer companies (such as IBM and DEC) and software vendors, the U.S. Department of Defense (DOD), and others are investing great sums in research and development to produce a *framework* to support an open, fully-integrated CASE environment. A reference model for CASE environment frameworks has been developed by the European Computer Manufacturers' Association (ECMA), and it has been submitted as a proposed standard to the National Institute of Standards and Technology (NIST). This model (figure 1) will be referred to as the NIST/ECMA Reference Model [1], but it is often informally called the "toaster model" because of its general appearance.

For the purpose of this paper, the NIST/ECMA reference model will be used to serve as an introduction to software engineering technology and as an illustration of how the different projects within the Software Technology Branch (STB) fit into the "big picture" of CASE environments (see figure 1). Therefore, it is useful to give a short description of the NIST/ECMA reference model for a CASE environment framework, but first some important software engineering concepts and some introductory background and definitions are presented.

2.1 Definition of Software Engineering

Software engineering may be defined as the planned process of producing well-structured, reliable, good-quality, maintainable software systems within reasonable budgets and time frames [2]. A few aspects of this definition need elaboration.

1) Software engineering is engineering. In particular, software engineering involves the application of systems engineering principles and techniques to the development of software systems. A methodical systems engineering approach is especially important in the case of large software systems which are typically extremely complex and require large teams of professionals to develop and maintain.
2) Software engineering involves both *product* and *process* [3]. A well-engineered software product is *documented* software that provides the services required by its users and which is maintainable, reliable, efficient, and provides an appropriate user interface. The software engineering process involves both technical and non-technical issues. As well as knowledge of specification, design and implementation techniques, software engineers must have some knowledge of human factors and software management [4].
3) Software engineering strives to produce cost-effective software systems via a cost-effective process. The software should be developed and maintained using appropriate cost, time, and personnel resources.

* The MITRE Corporation
** National Research Council Research Associate

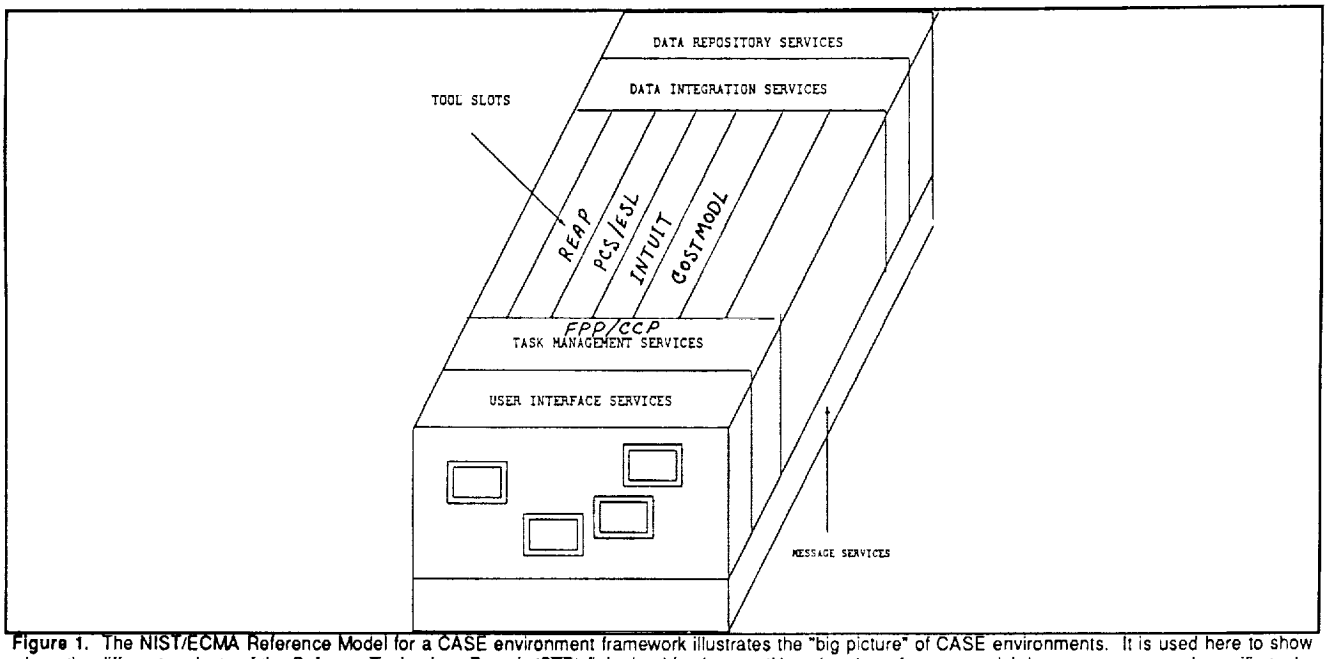

Figure 1. The NIST/ECMA Reference Model for a CASE environment framework illustrates the "big picture" of CASE environments. It is used here to show where the different projects of the Software Technology Branch (STB) fit in that big picture. (Note that the reference model does not recommend specific tools, and the STB tools shown here are not a part of the reference model [1].)

Learning how to be a good software engineer only begins with learning how to generate computer programs [5].

2.2 The Software Engineering Process

The phases through which a software system evolves during its lifetime, from the earliest exploratory phase in which the feasibility of the proposed system is explored to the phaseout stage in which the software system is discontinued, is referred to as the **software life cycle**. NASA's Software Management and Assurance Program (SMAP) has defined a standard, tailorable, Software Acquisition Life Cycle [6] that is composed of eight phases: concept and initiation, requirements, architectural (preliminary) design, detailed design, implementation, integration and testing, acceptance and delivery, and sustaining engineering (commonly called "maintenance") and operations.

Reports vary on the actual figures, but industry spends from 40% to 75% of its total hardware and software budget in the software maintenance phase alone [7]. This is because of changing requirements that arise from a changing environment and growing expectations. If software systems could be engineered (or reengineered) so that they are easier to understand and maintain, substantial savings might be realized during the maintenance phase. To help accomplish this goal, a specific software development **method** (i.e., a formal set of procedures or guidelines) should be employed during each life-cycle phase to produce the products of that phase. For example, there are three classes of methods applicable to the architectural and detailed design phases: top-down structured design, data-structure design, and object-oriented design [8]. Also, there is a need to capture metrics to evaluate how well a software development organization is performing, to track the quality of the software produced by the organization, and to determine whether changes result in actual improvements to the product and process.

The Software Engineering Institute (SEI) at Carnegie-Mellon University has developed a "Software Process Maturity" model (figure 2) for describing the maturity levels of software development organizations [9]. Most organizations have a very low maturity level (1), and only very few have a high maturity level (4 or 5). If the key problem areas at a given level are resolved by an organization, then that organization has progressed to the next higher maturity level in the model. Note the types of problem areas that typically exist in organizations at the various levels. In particular, automation (although potentially very valuable) cannot *by itself* improve an organization's software process maturity (at least, not until level 5 is attained).

2.3 Introduction to CASE

Computer-Aided Software Engineering (**CASE**) tools *assist* the software engineer in the application of software engineering methods. In the same sense that a word processor supports an author, a CASE tool provides technological support to a software engineer for drawing complex diagrams, for checking syntax and semantics, and in general for implementing a specific method efficiently. Clearly, a CASE tool cannot replace the software engineer anymore than a word processor can replace an author. Furthermore, CASE tools cannot assist in the software engineering process if such a process does not exist within a software development organization (i.e., if the organization's maturity level is 1). In such a situation, the organization must first adopt a software engineering process before it buys CASE technology; otherwise, the purchased CASE tools will rapidly become "shelfware."

2.4 Components of a CASE Environment

A **CASE environment** is an information system that provides support for software engineering. A CASE environment deals with information about software under development (such as project plans, requirements, designs, specifications, source code, and test data) and about an organization's software engineering process. Some alternative names for a CASE environment are Integrated Project Support Environment (IPSE) and Software Engineering Environment (SEE).

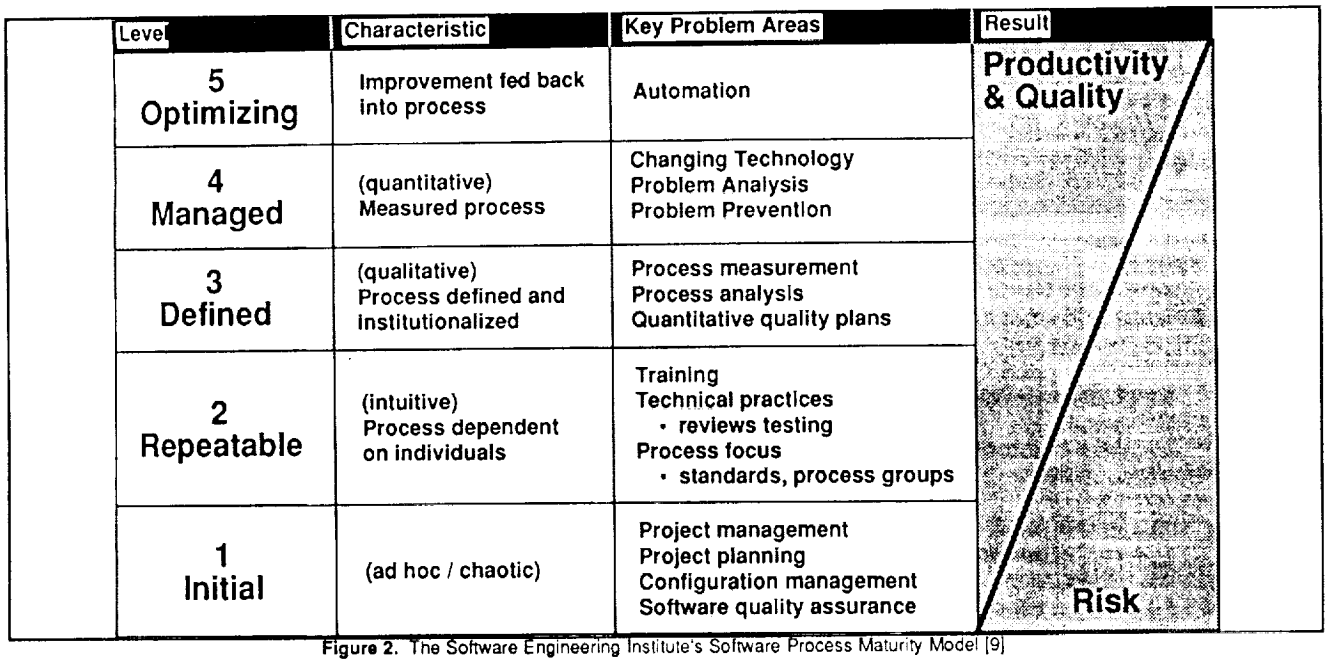

Figure 2. The Software Engineering Institute's Software Process Maturity Model [9]

A CASE environment consists of a relatively fixed set of core facilities, called the **environment framework**, and a set of facilities called **tools**, which are specialized for particular CASE environments and which are not available in all environments [1]. Tools that are **fully integrated** into the environment framework use the services provided by the environment framework extensively and may also use services provided by other tools. In contrast, **loosely integrated** tools use very few (if any) of the framework's services and do not use services provided by other tools.

2.5 The NIST/ECMA Reference Model

The tool slots shown in figure 1 represent the concept of sets of tools that can be readily integrated into a CASE environment framework in order to create special-purpose environments. The NIST/ECMA reference model does not advocate particular tools nor does it classify tools. However, two general categories of tools are classically defined: **vertical tools** (that are applicable during a single phase of the software development life cycle) and **horizontal tools** (that are applicable across the entire software life cycle). For example, a compiler and a code generator are vertical tools, but a project management tool is a horizontal tool.

Figure 1 is NOT meant to represent a functionally-layered model wherein one layer can only interface with adjoining layers. The reference model is simply based on grouping sets of framework services together in such a way that each grouping may be expected to be covered by existing or future standards. This aids in the definition and evolution of standards, a major goal of the NIST/ECMA reference model. This grouping of services also enables various kinds of integration to be discussed: **presentation integration** (user interface services), **control integration** (task management services plus the message services), and **data integration** (data repository services plus data integration services). The Appendix gives a brief description of each of the major groups of services.

Some of the aims of the NIST/ECMA reference model are:

1) provide a basis for describing, comparing, and contrasting existing and proposed environment frameworks;
2) provide a means for the smooth and coordinated evolution of future standards for environment frameworks;
3) address interoperability and integration of tools;
4) cover all framework aspects irrespective of implementation techniques or software development methods.

2.6 The Software Technology Branch's CASE Projects

As mentioned earlier, the NIST/ECMA reference model will be used in this paper to serve as an introduction to software engineering technology and as an illustration of how the different projects within the Software Technology Branch (STB) fit into the "big picture" of CASE environments (see figure 1). The rest of this paper discusses each of the STB's CASE projects.

- The STB is endeavoring to track the major CASE environment research and development efforts around the world, in order to get a better understanding of the big picture.
  - Section 3 gives the results of an analysis of today's CASE products and estimates the long-term prospects for CASE products.

- Other STB projects deal with tools that "plug into" the tool slots in figure 1.
  - The STB's Reengineering Application Project (REAP) is discussed in section 4.
  - The Advanced Software Development Workstation (ASDW) project is discussed in section 5, including descriptions of the reuse-oriented Parts Composition System (PCS) and Engineering Script Language (ESL) in 5.1, and the INTelligent User Interface development Tool (INTUIT) in 5.2.
  - The software development cost model (COSTMODL) project is discussed in section 6.

- The Framework Programmable Platform (FPP), a subtask of the ASDW project, is researching some of the environment framework's Task Management Services (see figure 1).

- The FPP subtask is developing a horizontal tool, called a Configurable Control Panel (CCP), for specifying, managing, and enforcing a model of the software development process. The FPP/CCP is discussed in section 5.3.

- Technology transfer of emerging CASE technology to NASA is the primary objective of all these STB projects.
  - The technology transfer process is discussed in section 7.

3. COMPUTER-AIDED SOFTWARE ENGINEERING: TODAY AND TOMORROW

As discussed in section 2, many major research and development programs are attempting to develop an open, fully-integrated CASE environment. Of course, the Software Technology Branch (STB) does not have the resources to develop such an environment on its own, nor is this necessarily desirable. However, the STB is attempting to identify those areas where CASE technology is approaching sufficient maturity that it can be transferred to NASA. Additionally, the STB is identifying those areas where CASE research and development show significant promise. In a few of these promising areas, NASA can leverage ongoing research and development work and thus ensure the development of the software technology that it will need to meet its own particular challenges.

A major project within the STB is the analysis of CASE technology today and the estimation of long-term prospects for CASE products. In a few cases, this analysis has even included very detailed, hands-on evaluation of prototype CASE environment frameworks, such as the United States Air Force's Software Life Cycle Support Environment [10,11].

3.1 Current Status of CASE Products

Today, there are hundreds of CASE vendors marketing tools. These vendors have heard the customers' demand for an integrated set of tools supporting software development activities across the full life cycle. The response has been in a variety of forms:

1) a vendor defines its set of tools as "full life cycle" (caveat emptor!);
2) a vendor buys or licenses another vendor's tools to create a more complete set of tools for the life cycle;
3) a vendor signs up as a business partner or program participant in one or more of the major corporations' CASE environment projects; or
4) a vendor becomes active in the standards meetings that are defining the various interfaces involved in a truly integrated CASE environment.

There are several messages to be understood in these different responses. The market consolidation indicates that no vendor, not even a major corporation, is willing to assume the total risk of investing in a completely proprietary CASE environment. Another message is that the educated consumer recognizes the dynamic nature of the technologies supporting CASE evolution. The STB is in the role of ensuring that JSC consumers do not buy CASE products which have no potential for evolution (upgrade).

Today's CASE consumer can buy the following:

1) tools that function on stand-alone, networked, mainframe-cooperative, or mainframe-dependent PCs and workstations;
2) tools that allow users on networked platforms to work with them simultaneously;
3) integrated tools from a single vendor that support the development of Management Information Systems (MIS) reasonably completely across the phases of the life cycle; and
4) tools for the development of Aerospace/Defense Engineering (ADE) systems. [These ADE tools are more limited and less integrated than their MIS counterparts. Some of these tools support DOD standards (e.g., MIL-STD 2167A). None support NASA standards (e.g., SMAP).]

The reasons for this situation are economic. 80-90% of the software developed today is business-oriented rather than engineering-oriented. Also the DOD has been funding software engineering methods and their implementation for 15-20 years. Nonetheless, NASA can leverage this work to its benefit.

For MIS applications, CASE technology can provide many reasonable solutions today. However, an open, fully-integrated CASE environment for the development of MIS does not yet exist, mainly because repository technology (for storing the megadata produced by CASE environments) is not yet mature enough.

There are some major deficiencies today in commercial CASE support for ADE systems. These deficiencies include:

1) quality of implementation of the method(s) to be used in a phase of the life cycle;
2) general lack of automated support for the generation of test scripts;
3) little or no assistance for reverse engineering of assembly code;
4) lack of adequate project management capabilities, including the lack of metrics which support project management; and
5) lack of a totally satisfactory notation for unambiguously expressing the design of a distributed system with multiple, asynchronous events.

There are also many positive signs in CASE products today for ADE and especially for real-time systems. The most significant of these is the capability to simulate the behavior of the system during the design stage from Program Design Language (PDL) generated on the basis of diagrammed behavior. Another significant capability of some CASE products is the ability to address the total system design. Large, complex systems are built to operate with hardware, software, and people. CASE tools for system co-design are becoming available in which components need not be specified as either hardware, software, or people functions in the early levels of the design.

3.2 Long-term Expectation for CASE Products

The unavailability of the special-purpose, truly-integrated CASE environments that are ideal for some applications will exist for at least another five years. Within five years, important interface standards will have been agreed upon and some environment frameworks will be robust enough for collections of compatible tools to be installed into a customized CASE environment. This will be a major step forward. However, integrable, plug-in horizontal tools will come later because they must cross the total life cycle development process in order to provide tailored documentation, project management, and configuration management and to measure the quality of the software engineers' work. The repository technology, especially the repository management system, must mature in order to provide robust support for horizontal tools.

Within ten years, mainframes may still be used for software execution but are unlikely to be used extensively for software development. Reliable, lower-cost, distributed approaches will likely be dominant. Some CASE environments may be globally distributed. Security issues for distributed systems will be

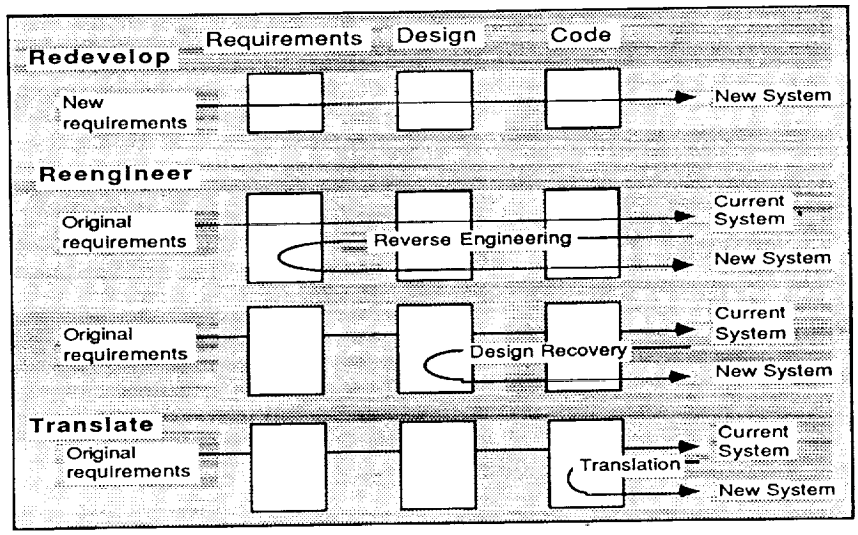

Figure 3. Alternative solutions to updating software systems to modern software engineering standards

resolved. Some CASE environments will have a rule base (or knowledge base) that can be configured with an organization's software development process in order to help manage the process. The project manager will assign responsibilities to his/her staff, constrain their access accordingly through rules, and automatically measure their skill and productivity against a variety of historical statistics. Reuse libraries will be well-categorized, accessible from a CASE environment, and the concept of synthesizing systems from "look-a-likes" will be supported. Many repetitive tasks will be automated. Much manual work will still be needed, but it will be concentrated in the systems engineering, specification, and design activities.

4. SOFTWARE REENGINEERING

The second important research project underway at the STB is the Reengineering Application Project (REAP). An environment (or, more accurately, an integrated tool set) for the reengineering of Fortran programs has been identified by the STB as a promising CASE area which, although requiring additional development, can leverage resources already developed for NASA.

Many Fortran systems developed in the 1960s and 1970s represent an enormous investment for the JSC. Today, these software systems are as crucial for the space program as they are expensive to use and maintain. This problem has led the STB to look for possible solutions and to consider the evolution path of these systems during the next 5 to 10 years. Among the three primary alternative solutions (i.e., complete redevelopment of the programs, code translation to a more modern language, and reengineering), reengineering appears to be the most promising path (figure 3). The goal of reengineering is to update a program to modern software engineering standards without losing required function and data and without losing the engineering knowledge embedded in the code. An integrated tool set for reengineering Fortran systems is needed to accomplish this goal efficiently. Currently, only a few commercial tools exist that aid in the reengineering of Fortran programs. The STB is not only investigating these tools but is also researching and developing the capabilities needed to completely support the reengineering of Fortran systems.

4.1 Definitions

Reengineering is the combination of the reverse engineering of a working software system followed by the forward engineering of a new system based on the results of the reverse engineering.

Forward engineering is the standard process of developing software from requirements. It is composed of many life cycle phases such as requirements, architectural design, detailed design, coding, and testing.

Reverse engineering is the reverse of forward engineering. It is the process of starting with existing code and going backward through the software development life cycle. Life cycle products are obtained by abstracting out only the essential information and hiding the non-essential details at each reverse step.

How far to go backward in the reverse engineering process before it is stopped and forward engineering begins is a critical question and involves trade offs. It is important to understand all of what the program does, all of the information it handles, and the control flow. In other words, the reverse engineering process must be carried far enough back to understand *what* the "as is" program is.

Reverse engineering is referred to as **design recovery** when the reverse engineering process stops at the recovery of the design rather than proceeding on to a higher level of abstraction to include the recovery of the requirements. The basic process of design recovery involves recovery of information about the code modules and the data structures in an existing program. This information can support the programmer/analyst(s) who is either maintaining a large, unfamiliar Fortran program, upgrading it for maintainability, or converting it to another target language.

Of course, a better job of redesigning a program can be accomplished when the reverse engineering process is carried beyond design recovery to **requirements recovery**. However,

requirements recovery is difficult and requires higher levels of domain knowledge to do the abstractions. The *whys* of the requirements, design, and implementation can only be provided by someone very familiar with the program and the domain. This level of expertise is often very difficult to find and to have dedicated to the reengineering process.

4.2 Reengineering Strategy

The STB is proposing standards, methods, and an integrated reengineering tool set [12,13] based upon the significant set of tools built to develop and maintain Fortran programs for the Space Shuttle [14,15]. The proposed tool set must support these standards and methods even in areas where the language definition and compilers do not enforce good software engineering practices. The intent is to get an integrated tool set out into use in JSC's maintenance community to provide support for upgrading Fortran programs in terms of maintainability in the near term, and then to extend the functionality of the tool set in response to feedback from the programmers/analysts. Later versions of the integrated tool set may have extensions to handle programs written in C, Ada, or even HAL/S, according to requests from the user community.

The STB has defined new standards for its Fortran programs [12]. These new standards are added to previously defined standards for Fortran programs which specified coding standards, in-line documentation standards, global data structure standards, and unique data naming conventions. The new standards produce a Fortran program with good software engineering structures such as those found in Ada. These new Fortran standards address documentation, longer variable names, modern control flow structures, grouping of subroutines into virtual packages, data structuring, and separation of input/output processing from the principal functionality provided by the program.

The reengineering methods developed by the STB are aimed at progressively converting a Fortran program into more maintainable states [12]. They define the way to convert an arbitrary Fortran program to the STB's previously defined Fortran standards, then to the new standards, and finally to a target language that embeds software engineering principles (such as Ada). These reengineering methods define the steps and the skills required and give guidelines on how far to reverse engineer before deciding to rebuild.

The STB's proposed integrated reengineering tool set [12] will support the above standards and methods. A preliminary tool set has been developed that supports the maintenance of Fortran programs and that aids in the initial analysis phase of reengineering (to Fortran 90, C, or Ada). It contains modified versions of the tools used to support the current STB Fortran standards, plus commercial off-the-shelf (COTS) tools, and additional custom-built tools. The tools are integrated at the front end by a user interface (i.e., presentation integration) and behind the screen by two logical databases, an inter-tool database and a source code database. However, COTS tools cannot be integrated seamlessly into the tool set at this time. This tool set will not completely automate the reengineering process since much reengineering work must still be done by a programmer/analyst. However, as an experience base is accrued in design recovery, knowledge-based capabilities can be added to assist the programmer/analyst even further.

5. THE ADVANCED SOFTWARE DEVELOPMENT WORKSTATION PROJECT

As discussed in section 2, the NIST/ECMA model of a CASE environment framework does not recommend specific tools nor does it classify them. The STB's analysis of CASE product status and direction indicates that a number of good tools are available, but there are some important capabilities and tools that still require further research and development. The Advanced Software Development Workstation (ASDW) project is researching and developing specific types of advanced technology and tools that an advanced workstation for software development should provide [16]. One of these tools is a Parts Composition System (PCS) for developing applications from reusable software parts, using knowledge-based technology. Another tool being developed is the INTelligent User Interface development Tool (INTUIT). A third tool, a Configurable Control Panel (CCP) for an integrated CASE environment, is being developed under the Framework Programmable Platform (FPP) subtask of the ASDW project. The CCP will be a horizontal tool for managing and enforcing a (locally configurable) model of the software development process.

5.1 Parts Composition System and Engineering Script Language

Many researchers think that software reuse is a way to significantly improve the quality of applications and the productivity of application developers. The ASDW's Parts Composition System (PCS) is a set of tools for developing applications from reusable software parts. Before describing the PCS, however, one aspect of the concept of compositional reusability demands a brief explanation.

A library of procedures (or, more generically, primitives) containing the reusable software parts must be available before they can be put together to form a complete application; i.e., the components must exist before they can be assembled. It is very important to understand that not just any procedure can qualify as a *reusable* software part. Reusable primitives must be specifically designed for reuse. They must be optimally designed to strike a balance between the desire for general applicability and the need for applicability to a given class of problems. In other words, reusable parts must be carefully engineered so that they can be used throughout an explicitly specified domain.

The development, organization, and maintenance of these domain-specific libraries of reusable parts is primarily the responsibility of the software development engineers and not the job of the application developers, who may be aerospace engineers with minimal programming experience. The software development engineers receive part specifications from the application developers and provide implementations to populate the required libraries. If well managed, this separation of roles helps to limit the amount of domain expertise that the software developer must have and also the amount of programming experience that the application developer must have.

Once an application developer has selected the most appropriate domain-specific library of parts, he/she invokes a PCS tool called the Engineering Script Language (ESL) editor. This is a graphical editor that allows the application developer to create, modify, store, and retrieve graphs that represent applications. The graphs show the structure of an application (i.e., how it is made from its component parts) and what data and controls flow between the components. The components are depicted by boxes called nodes, and the data and controls are shown as arrows linking the nodes. Other structures allow for merging and replicating links and for including looping and branching logic. Each component (box) is either a primitive from the library or a subgraph, which makes possible hierarchical decomposition.

Once the graphs representing an application are completed, the application developer will invoke menu commands to validate the graph system and to generate the required code in some high-order language, such as Ada. The generated code, in the form of a main program and subprograms, will then be ready to be compiled and linked with the object code of the primitives from the

domain-specific library. Alternatively, source code templates (such as Ada generics or even main programs with certain parameters that must be initialized before compilation) might be generated, if required.

The internal representation and storage of the ESL graphs, the semantic interpretation and validation of the graphs, and the generation of code in a high-order language will be done using knowledge-based technology. To date, specifications have been developed for the ESL system. Implementation of the first ESL prototype is underway.

5.2 INTelligent User Interface Tool

A good user interface is critical to the successful use of a complex scientific application such as a space flight simulation, which typically involves very large sets of input data. Even an expert user may expend substantial effort to introduce the right data in the right manner. An Intelligent User Interface (IUI) uses knowledge-based technology to provide the user with the capability to easily prepare the input data without requiring prior extensive knowledge of the underlying software. An IUI is also commonly called a Knowledge-Based Front-End (KBFE). INTUIT (INTelligent User Interface development Tool) is a generic IUI shell that a knowledge engineer configures for a specific application by adding a knowledge base that includes input variable names which are immediately understandable by the users, the range of permissible data values, the structure and format of the data sets, and rules for error and consistency checking. The current knowledge representation scheme used within an INTUIT knowledge base is fully described in [17]. Many of the same subsystems required by a PCS are also required by INTUIT, which may therefore be considered to be a "PCS for input data sets." In fact, INTUIT is a PCS subshell.

INTUIT's user-friendly interface relies on the Transportable Applications Environment Plus (TAE+) [18,19] developed by NASA/Goddard Space Flight Center to provide a graphical windowing environment with a mouse and icons. TAE+ is a very powerful graphical interface development tool, built as a layer on top of X-Windows (from M.I.T.). The graphical interfaces built using TAE+ are very portable. INTUIT provides standard panels for the user to browse and modify objects in the knowledge base. In addition, INTUIT allows the knowledge engineer to design "custom forms" or input screens using the TAE+ Workbench and to describe these forms via schemas in the knowledge base. This capability gives the knowledge engineer a way to specify good data organization and to enforce it, as well as providing a modern user-friendly interface with point and click capabilities.

The INTUIT shell was used to develop a KBFE for Space Vehicle Dynamics Simulation (SVDS), a computer program currently used at JSC for designing the trajectory and flight plans for Space Shuttle missions. The SVDS application called Ground Simulation (GNDSIM) [20] was selected for KBFE development, and an INTUIT knowledge base was built for it. Flight planners use GNDSIM to verify and refine the sequence of maneuvers required to accomplish a rendezvous. A thorough discussion of the development and testing of this KBFE for GNDSIM is presented in [21], but the results can be summarized as follows. All the users who participated in the tests were very satisfied with the KBFE. Building an input data stream with the KBFE proved to require from one-half to one-fifth the time needed using the current interface. As a result of these tests, specific enhancements to INTUIT are planned. The development of KBFEs for other tools used by the flight designers is also being considered.

5.3 Framework Programmable Platform

The Framework Programmable Platform (FPP) subtask of the ASDW project is researching some of the issues involved in building an integrated CASE environment. The current focus of the FPP is the management and control of the software development and maintenance processes -- crucial factors in the success or failure of any large software system. In the NIST/ECMA model (figure 1), the services that the FPP is focusing on are part of the Task Management Services. Specifically, the FPP subtask is developing a horizontal tool, called a Configurable Control Panel (CCP), for specifying, managing, and enforcing a model of the software development process.

The CCP, which captures an organization's experience and best-practice rules of software development, will be configured by a system administrator/manager of the software development organization and will define for each step of the software development process [22]:

1) the minimum tasks that must be performed to complete the step,
2) the methods, tools, and computer resources available to projects at the local site,
3) the personnel roles that can be assigned to project personnel, and
4) the configuration control method(s) to be applied to the artifacts and products developed.

As it is configurable, the CCP can be tailored according to the different management needs of different types of projects. For example, a research project may allow unlimited access to all resources by all members of the small group of professional staff assigned to do the research, but a project to develop a million lines of mission critical code for the Space Station that has 100 programmers working simultaneously would probably need to strictly control, not only access, but also the tools to be used for each step of the process.

Today, the CCP is still in the design phase. It will utilize both conventional and knowledge-based technology; a knowledge-based system is planned to help configure the CCP. The IDEF3 method for process description, a graphical method developed for the U. S. Air Force, will be used to specify the steps in the software development process. The user interface will display these IDEF3 process diagrams as a guide for the user; shading or coloring will be used to indicate which activities have been completed and which ones the user is not authorized to perform (and will not be allowed to access). Buttons and menus will be used to help choose the best tool(s) to perform each task and to list the artifacts which must be produced before the task is considered complete. The user interface will also display a matrix of software development perspectives versus task focus, based upon concepts of Information Systems Architecture suggested by J. Zachman [23].

6. COST MODELING

Organizations such as NASA that make large expenditures for the development of software can realize significant benefits by including a good model for the estimation of software costs in their set of software engineering tools. COSTMODL was designed to be an in-house tool for the estimation of NASA software development costs, but it can be used by other organizations as well [24]. It provides project managers with an automated tool that permits them to obtain credible development cost estimations without having to rely on software costing specialists and to check estimates provided in contract proposals.

6.1 Characteristics of COSTMODL

COSTMODL is a flexible tool for estimating the effort and schedule required to develop a software product of a given size. Five separate cost estimation models have been incorporated into

COSTMODL. These are the KISS (Keep It Simple Stupid) model, the Basic, Intermediate and Ada COCOMO models, and the Incremental Development Model. This choice of models covers the spectrum of project complexities from small, relatively simple projects to very large projects developed in an Ada software engineering environment and delivered in a series of separate but related increments. All of the parameters defining each of the models are accessible to the user. The basic estimating equations can be calibrated to the user's software development environment and type of products, and the set of factors which influence software development costs can be redefined.

6.2 Definition of the Models Used

KISS. The KISS model is a simplified linear estimating model which was developed using productivity data derived from past NASA software development projects. In addition to the anticipated product size, the user specifies the project criticality, type of software, type of language, development team productivity and an Ada Productivity factor.

COCOMO. The COCOMO model, originated by Dr. Barry Boehm, is the most widely used cost estimation model due to its proven performance over the years, and to the fact that it is in the public domain, allowing one to tailor it to its own environment.It consists of a family of models.
The Basic COCOMO model is a very simple approximation which accepts only the size of the total product to perform the evaluation. It is good for quick early rough order of magnitude estimates.
The Intermediate COCOMO model is a more sophisticated model which provides a mechanism whereby the user can specify a set of factors to account for differences in hardware constraints, personnel quality and experience, use of modern tools and techniques and other project attributes known to have a significant influence on software costs.
The Ada COCOMO model is an extension to the Intermediate model which attempts to quantify the changes in programmer productivity resulting from the use of good software engineering practices.
Incremental Development model. The incremental development model provides for the division of a total project into separate stand-alone deliveries. It computes the individual increment effort and schedule and the total project effort and schedule, taking into account the amount of rework required on the earlier increments to accommodate the later increments.

6.3 Status

COSTMODL is now distributed for NASA by Computer Sciences Corporation (713-280-2233) and is being prepared for distribution through the Computer Software Management and Information Center (COSMIC) at the University of Georgia. It is available free to government agencies and their contractors. It is being used at several hundred government, military and contractor sites, and has been selected as the standard cost estimating tool for NASA's Space Station Freedom Program (SSFP).

7. TECHNOLOGY TRANSFER

Technology transfer of emerging CASE technology to NASA is the primary objective of all of the STB's CASE projects. Technology transfer can occur in many different ways, including the direct use of tools developed by the STB (such as COSTMODL), the adoption of new methods and technology identified by the STB (such as repository technology when it matures), and the purchase of appropriate commercial off-the-shelf (COTS) tools. This section discusses a process being developed by the STB to help insert appropriate CASE technology at JSC.

What techniques should be used to select and insert CASE technology? At first, the STB set a goal of collecting information about existing CASE products and characterizing those products in order to provide a CASE tool consulting service for the JSC community. The motivation for establishing such a service was the complex nature of CASE and the confusing status of the CASE tool market. The sheer number of available CASE tools and the rapid rate of change of the CASE market, coupled with unrealistic consumer expectations of what CASE tools can do, have led to some exaggerated claims about CASE and, consequently, to some disappointed consumers.

Rather quickly, the scope of this CASE consulting service was expanded to a software engineering consulting service. This is because CASE tools cannot produce magical results (despite the claims of many vendors); i.e., CASE tools can only *assist* in the software engineering process. A software development organization must still employ people to do good software engineering and must still have a well-managed, repeatable, and *explicit* software development process. If a disciplined software engineering process does not exist within an organization, then that organization must adopt one, and this will likely imply a change in its way of doing business.

How do organizations successfully introduce new ways of doing business? Technology insertion, in particular, seems to be successful only when key people within the organization (called change agents) actively participate. Change agents tend to be knowledgeable persons in middle management, recognized for their abilities and credibility by both upper and lower management.

In order to make sound CASE recommendations and to improve the chances of achieving CASE technology insertion, the STB is developing a CASE Selection and Insertion Process [25,26]. In simplest terms, there are five basic activities that occur during the process: characterize the organization's culture; characterize the software systems produced; identify improvements to the organization's software engineering process; identify candidate tools and environments; and develop a technology insertion plan.

As a consequence of an organization's request to the STB for assistance in the choice of automation techniques for their software development, there is an initial survey of the situation and problems faced by the organization. Meetings are held with managers who identify persons to be interviewed for information and to support the process.

The characterization of the "culture" of the organization has some similarity to the Software Engineering Institute's Software Maturity Assessment. It is important to have an in-depth understanding of how the organization currently does business, i.e., what the characteristics of the software development process in use by the organization are, and which software engineering methods and notations are already understood or preferred. If there is no documented process nor consistent methods and if there is no tool which supports the current way the organization works, a lot of change is in order.

The type of applications being developed by the organization must be determined. This is important because different CASE tools are required for different types of applications. In the simplest model, all applications have three components to them: data, function, and control. An MIS system is data-focused, and so the data modeling capabilities of the CASE tool are of primary importance. On the other hand, an engineering application with real-time characteristics has design concerns primarily with function and control, and so the CASE tool must be able to model the behavior of the system appropriately. Some other important characteristics are the database and language to be used in the applications being developed, especially if the CASE tool will be required to do code generation.

The identification of possible improvements to the organization's software engineering process is a critical portion of the technology insertion work. Potential improvements should be frequently discussed with the organization and feedback obtained. Feedback from the organization is essential because the STB is trying to provide useful advice that will solve some of the organization's real needs and is not trying to dictate policy.

Once the organization and its applications are characterized and analyzed, a mapping is made between these results and the characteristics of CASE tools. A set of tentative requirements emerges which is used to filter potential candidate tools from the universe of known tools. The weighting factors that should be applied to particular requirements are determined. These weights are affected by trade-off considerations such as the long-term benefits versus coststo the software development organization to change its current process or methods or to obtain different or upgraded hardware, if necessary.

The universe of known tools that is referenced in order to identify candidate tools consists of three collections. One is a manual file collection of some 90 vendors' brochures, demonstration disks, and commentary thereon kept in file drawers. A second is a public domain database from the Air Force's Software Technology Support Center (STSC) which contains commentary on tools tested by or used by contributors to the database. (Users are expected to update the database with local commentary and return it to STSC on a quarterly basis.) The third collection of information is a commercial database from P-Cube, Inc. which allows the user to query for tools on certain prescribed capabilities. (P-Cube sends monthly updates.) The electronic databases are maintained in the STB's Laboratory.

Armed with a short list of candidate tools, the STB presents its findings to the software development organization, obtains additional feedback, and repeats the requirements definition and tool filtering as many times as is appropriate. However, the organization must obtain hands-on experience with the candidates prior to *their own* final selection. To conclude the consulting process, the STB presents the completed technology insertion plan. The STB provides any final suggestions for improvements to the software engineering process and recommends the necessary training for the software engineering methods to be used, the new hardware, and the CASE tool itself. CASE tools that have extensive capabilities for use with complex systems are non-trivial to use. To avoid failure, training for new users and a continuing program of training are essential.

Currently, the CASE Selection and Insertion Process is being applied within two other branches of the Information Systems Directorate at JSC. Support for MIS applications has been targeted for this initial testing of the Process. Preliminary findings, recommendations, and technology insertion plans have been presented to these two organizations. The STB is currently reviewing the results of this initial testing of the Process in order to revise it accordingly. Discussions are also in progress between the Mission Operations Directorate and the STB regarding the need for CASE support.

8. PLANS FOR THE FUTURE

The STB will continue to track the progress of software engineering technology, especially in CASE environment frameworks, standards, and methods. Development and refinement of the REAP environment, the ASDW technology and tools, and COSTMODL is continuing. A strong synergistic potential exists between the REAP and ASDW projects, especially in the area of domain-specific architectures, a research area aimed at identifying standard architectures for various domains in order to permit the development (or reengineering) of standard, reusable parts for each of these architectures. The potential for collaboration also exists between the ASDW project and two other STB projects, the Intelligent Computer-Aided Training (ICAT) project and the Task Analysis/Rule GEnerating Tool (TARGET) knowledge acquisition project. These areas of cooperation will be actively pursued. Finally, the CASE Selection and Insertion Process will be refined and extended, and the STB will examine the feasibility of setting up a CASE product demonstration facility to provide an opportunity for potential users to assess the features of a choice of CASE tools and environments.

9. SUMMARY AND CONCLUSION

This paper has discussed ongoing work within the Software Technology Branch (STB) to identify and develop some of the Computer-Aided Software Engineering (CASE) technology that NASA will need to meet the software development and maintenance challenges of the future. The main goals of this work are to improve the productivity of software developers, users, and maintainers and to improve the quality of software systems. Specifically, the following STB projects have been discussed in this paper:

- The STB has identified, evaluated, and characterized leading CASE products, and projections of the long-term prospects and direction of CASE have also been made. (Sec. 3)
- The Reengineering Environment Application Project (REAP) has proposed Fortran standards, reengineering methods, and the types of tools required in an ideal, integrated, reengineering tool set. A preliminary environment to support maintenance and reengineering analysis has been developed. (Sec. 4)
- The Advanced Software Development Workstation (ASDW) project is researching and developing specific types of advanced technology and tools that an advanced workstation for software development should provide. A Parts Composition System (PCS), which includes an Engineering Script Language (ESL) editor, for developing applications from reusable software parts is being developed. Testing of a PCS subshell, the INTelligent User Interface development Tool (INTUIT), has been recently completed. The Framework Programmable Platform (FPP) subtask is developing a Configurable Control Panel (CCP) for an integrated CASE environment, which will enforce an organization-specified model of the software engineering process. (Sec. 5)
- COSTMODL is a mature, flexible tool for estimating the effort and schedule required to develop a software product of a given size. It is currently being used at several hundred government and contractor sites.(Sec.6)
- Technology transfer of emerging CASE technology to NASA is the primary objective of all of the STB's CASE projects. The STB has developed a CASE Selection and Insertion Process and is currently testing it. (Sec. 7)

In order to meet the challenges of tomorrow's space program, there is a need to increase the software process maturity level of NASA organizations and contractors and to enforce good software engineering principles. The STB believes that the use of CASE tools will be a major benefit to NASA in the coming years. It is anticipated that the technology being identified and developed by the STB, and by many other NASA organizations as well, will play a strategic role in future NASA software development projects.

10. ACKNOWLEDGMENT

We acknowledge the National Research Council's sponsorship (through its Research Associateship Program at the Johnson Space Center) of M. E. Izygon, one of the authors of this paper.

11. REFERENCES

1. Earl, Anthony, "A Reference Model for Computer Assisted Software Engineering Environment Frameworks," Ver. 4.0 ECMA/TC33/TGRM/90/016, Hewlett-Packard Labs., Software Environments Group, Bristol BS12 6QZ, England, Aug., 1990.

2. Simmons, G. L., "What is Software Engineering?" ISBN 0-85012-612-6, NCC Publications, 1987, as quoted in Earl [1], p. 1.

3. Boehm, Barry W., "The Goals of Software Engineering," SOFTWARE ENGINEERING ECONOMICS, Prentice-Hall, Englewood Cliffs, NJ, 1981, p. 23.

4. Sommerville, Ian, "Introduction," SOFTWARE ENGINEERING, 3rd Ed., Addison-Wesley, Menlo Park, CA, 1989, p. 20.

5. Boehm [3], p. 16.

6. NASA, Office of Safety, Reliability, Maintainability, and Quality Assurance, Software Management and Assurance Program (SMAP), "NASA Software Acquisition Life Cycle," Ver. 4.0, NASA, Washington, D. C., 1989.

7. Booch, Grady, "The Software Crisis," and "Software Engineering," SOFTWARE ENGINEERING WITH ADA, 2nd Ed., Benjamin/Cummings, Menlo Park, CA, 1987, pp. 8 &29.

8. Booch [7], pp. 36-37.

9. Humphrey, Watts, "CASE Planning and the Software Process," CMU/SEI-89-TR-26, Carnegie-Mellon University/Software Engineering Institute, Pittsburgh, PA, 1989.

10. Rogers, Kathy L., "Software Engineering Environment Frameworks, Vol. I: Evaluation Method," MTR-91W00048-01, The MITRE Corp., Houston, TX, April, 1991.

11. Rogers, Kathy L., "Software Engineering Environment Frameworks, Vol. II: Evaluation of the Software Life Cycle Support Environment," MTR-91W00048-02, The MITRE Corp., Houston, TX, May, 1991.

12. Fridge III, Ernest, Braley, Dennis, & Plumb, Allan, "Maintenance Strategies for Design Recovery and Reengineering," Vols. 1-4, NASA Johnson Space Center, Houston, TX, June, 1990.

13. Plumb, Allan, & George, Vivian, "A Method for Conversion of Fortran Programs," Barrios Technology, Inc., Houston, TX, March, 1990.

14. Braley, Dennis, "Automated Software Documentation Techniques," NASA Johnson Space Center, Houston, TX, April, 1986.

15. Braley, Dennis, "Software Development and Maintenance Aids Catalog," JSC-22349 (86-FM-27), NASA Johnson Space Center, Houston, TX, October, 1986.

16. Fridge III, E. M. and Pitman, C. L., "The Advanced Software Development Workstation Project," SOAR90, Albuquerque, NM, June 26-28, 1990.

17. Pitman, C.L., Izygon, M.E., Ralston, E.W., FridgeIII, E.M., and Allen, B.P., "Intelligent Interfaces For Complex Software," Fourth International Conference on Industrial & Engineering Applications of Artificial Intelligence and Expert Systems, Kauai, Hawaii, June 2-5, 1991.

18. NASA Goddard Space Flight Center, "Transportable Applications Environment Plus," Ver. 4.1, GSC-13275 with documentation, COSMIC, The University of Georgia, Athens, GA, Jan., 1990. Ver. 5.1 (MOTIF), GSC-13448, released May, 1991.

19. Szczur, Marti, "Transportable Applications Environment (TAE) Plus: A NASA Tool Used To Develop And Manage Graphical User Interfaces," SOAR91, Houston, TX, July 9-11, 1991.

20. UNISYS Houston Operations, GNDSIM User's Guide.

21. Izygon, M.E., and Pitman, C.L., "A Knowledge Based Front-end for a Complex Space Flight Simulation Program," 1991 Summer Computer Simulation Conference, Baltimore, MD, July 22-24, 1991.

22. Mayer, R. J., Blinn, T. M., and Mayer, P. S. D., "Framework Programmable Platform for the Advanced Software Development Workstation: Concept of Operations Document," Report to NASA and University of Houston-Clear Lake by Knowledge Based Systems Inc. under subcontract SE.37, NCC9-16, Sept. 18, 1990.

23. Zachman, J., "A Framework For Information Systems Architecture," IBM Systems Journal., Armonk, NY, Vol. 26., No. 3, Sept., 1987, pp. 276-292.

24. Roush, G. B., "COSTMODL User's Guide," Preliminary Draft, NASA, Johnson Space Center, Software Technology Branch/PT4, Houston, TX, March, 1991.

25. Erb, D. M., "Evaluation and Selection of CASE Products," Presentation to Software Technology Branch, The MITRE Corp., Houston, TX, Nov. 19, 1990.

26. Pitman, C., Erb, D., Rogers, K., and Dorofee, A., "The CASE Selection Process: Essential Data and Conceptual Example," Ver. 1.0, NASA, Johnson Space Center, Software Technology Branch/PT4, Houston, TX, May 16, 1991.

27. Open Systems Foundation (OSF), "OSF User Environment Component: Decision Rationale Document," OSF, Jan. 11, 1989, as quoted in [1], pp. 51-52.

APPENDIX

This appendix gives a brief description of each of the major groups of services in the NIST/ECMA reference model [1] shown in figure 1.

Data Repository Services. The maintenance, management, and naming of data entities or objects and the relationships among them is the general purpose of the data repository. Basic support for process execution and control is also addressed here along with a location service to support physical distribution of data and processes. The classes of services in this group are: data storage, relationship, name, location, data transaction, concurrency, process support, archive, and backup.

Data Integration Services. The data integration services enhance the data repository services by providing higher-level semantics and operations with which to handle the data stored in the repository. The classes of services in this group are: version, configuration, query, metadata, state monitoring, sub-environment, and data interchange.

Tools. The entire set of environment framework services exist partly to support one another, but mainly to support tools that provide assistance for particular forms and methods of software engineering. Tools plug into the CASE environment framework. In addition to fully integrated tools that extensively use an environment framework's services, the tool slots in an environment framework may also provide an **encapsulation service**. This service is used when a tool exists (but was not written to make use of any of a particular environment framework's services) and is made to work in the environment framework by surrounding the tool with software that acts as a layer between the tool and the framework. The encapsulated tool fits into the framework without modification, but it uses very few (if any) of the framework's services.

Task Management Services. These services, sometimes called **software process management services**, allow the user to deal with major tasks as opposed to accomplishing each job by a tedious series of invocations on individual tools. The classes of services in this group are: task definition, task execution, task transaction, task history, event monitoring, audit and accounting, and role management.

Message Services. These services provide standard, two-way communications between services, between tools, and between tools and services. The classes of services in this group are: message delivery and tool registration.

User Interface Services. The importance of separating the presentation of functionality from the provision of functionality is widely recognized, and a consistent user interface service may be adopted for a complete environment framework. However, because of the complexity and generality of user interface issues, the NIST/ECMA model does not include a user-interface reference model of its own. It does summarize an existing user-interface reference model, which it recommends as a good starting point for discussions and which may be described as the (relatively) familiar "X-Windows layered model" [27].

Security Services and Framework Administration and Configuration Services. These last two groups of services are not shown in figure 1. Security services affect all of the other groups and cross many of the logical group boundaries represented in figure 1. Three classes of security services are: security information, security control, and security monitoring. Framework administration and configuration services are vital to a CASE environment, but the reference model (conceptually) views these services as customizations of the other framework services already discussed.

The NIST/ECMA reference model also specifies a way to completely describe each of the services mentioned above, in order to ensure compatibility, comparability, and precision of descriptions. In their presentation of the reference model, the following **dimensions** are used as sub-headings for the descriptions of particular services: Justification, Conceptual, Degree of Understanding, Data dealt with, Types, Operations, Rules, Metadata, Instances, External, Internal, and Related Services [1]. The scope of this paper does not permit further discussion of these dimensions.