US Army Corps
of Engineers
Construction Engineering
Research Laboratories

USACERL Technical Report FF-94/25
June 1994

Knowledge Worker Platform Analysis

by
Edward J. Japel
Melody M. Moore
Wayne Schmidt
Spencer Rugaber
Hernan Astudillo
and Scott Maxwell

Many Army personnel can be classified as *knowledge workers*—people who produce not tangible products, but some form of processed or enhanced information. Most Army knowledge workers depend on computer processing to complete their tasks efficiently. However, those tasks are often complicated by the many computer platforms and software packages used to contain and convey needed information.

The U.S. Army Construction Engineering Research Laboratories (USACERL) has been conducting ongoing research into the problems of information access and management for knowledge workers, with the ultimate goal of developing a comprehensive performance support environment for knowledge workers. The Knowledge Worker System (KWS) is a prototype scheduling program designed to help knowledge workers organize and coordinate their work by storing task scheduling information in a centralized data base. KWS tracks scheduled events, provides a list of completed events, and outlines the steps necessary to complete forthcoming tasks. This study examined the feasibility of converting KWS to an "open systems" technology to make the program compatible with a number of different platforms. Current marketability of language tools, graphical user interface (GUI) tools, and operating systems were investigated for compliance with government and open systems standards. Strategic plans were devised for KWS conversion.

USER EVALUATION OF REPORT

REFERENCE: USACERL Technical Report FF-94/25, *Knowledge Worker Platform Analysis*

Please take a few minutes to answer the questions below, tear out this sheet, and return it to USACERL. As user of this report, your customer comments will provide USACERL with information essential for improving future reports.

1. Does this report satisfy a need? (Comment on purpose, related project, or other area of interest for which report will be used.)

_____

_____

_____

2. How, specifically, is the report being used? (Information source, design data or procedure, management procedure, source of ideas, etc.)

_____

_____

3. Has the information in this report led to any quantitative savings as far as manhours/contract dollars saved, operating costs avoided, efficiencies achieved, etc.? If so, please elaborate.

_____

_____

4. What is your evaluation of this report in the following areas?

a. Presentation: _____

b. Completeness: _____

c. Easy to Understand: _____

d. Easy to Implement: _____

e. Adequate Reference Material: _____

f. Relates to Area of Interest: _____

g. Did the report meet your expectations? _____

h. Does the report raise unanswered questions? _____

i. General Comments. (Indicate what you think should be changed to make this report and future reports of this type more responsive to your needs, more usable, improve readability, etc.)

5. If you would like to be contacted by the personnel who prepared this report to raise specific questions or discuss the topic, please fill in the following information.

Name: _____

Telephone Number: _____

Organization Address: _____

6. Please mail the completed form to:

Department of the Army
CONSTRUCTION ENGINEERING RESEARCH LABORATORIES
ATTN: CECER-IMT
P.O. Box 9005
Champaign, IL 61826-9005

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 1994	Final

4. TITLE AND SUBTITLE
Knowledge Worker Platform Analysis

5. FUNDING NUMBERS
MIPR
DLAH-92-ZRM-206

6. AUTHOR(S)
Edward J. Japel, Melody M. Moore, Wayne Schmidt, Spencer Rugaber, Hernan Astudillo, and Scott Maxwell

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Construction Engineering Research Laboratories (USACERL)
P.O. Box 9005
Champaign, IL 61826-9005

8. PERFORMING ORGANIZATION REPORT NUMBER
FF-94/25

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Logistics Agency
ATTN: DLA-ZI
Room 3B527 Cameron Station
Alexandria, VA 22304-6100

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
Copies are available from the National Technical Information Service, 5285 Port Royal Road, Springfield, VA 22161.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

Many Army personnel can be classified as *knowledge workers*—people who produce not tangible products, but some form of processed or enhanced information. Most Army knowledge workers depend on computer processing to complete their tasks efficiently. However, those tasks are often complicated by the many computer platforms and software packages used to contain and convey needed information.

The U.S. Army Construction Engineering Research Laboratories (USACERL) has been conducting ongoing research into the problems of information access and management for knowledge workers, with the ultimate goal of developing a comprehensive performance support environment for knowledge workers. The Knowledge Worker System (KWS) is a prototype scheduling program designed to help knowledge workers organize and coordinate their work by storing task scheduling information in a centralized data base. KWS tracks scheduled events, provides a list of completed events, and outlines the steps necessary to complete forthcoming tasks. This study examined the feasibility of converting KWS to an "open systems" technology to make the program compatible with a number of different platforms. Current marketability of language tools, graphical user interface (GUI) tools, and operating systems were investigated for compliance with government and open systems standards. Strategic plans were devised for KWS conversion.

14. SUBJECT TERMS
Knowledge Worker System (KWS)
open systems technology
Ada (computer program language)
task scheduling information

15. NUMBER OF PAGES
44

16. PRICE CODE

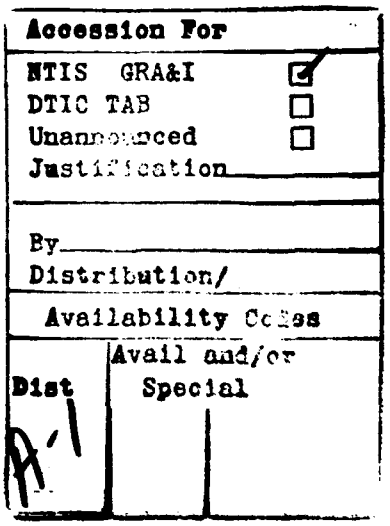
17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	SAR

Foreword

This study was conducted for Defense Logistics Agency under Military Interdepartmental Purchase Request (MIPR) No. DLAH-92-ZRM-206. The technical monitor was Harriet Riofrio, DLA-ZI.

The work was performed by Facility Management Division (FF), Infrastructure Laboratory (FL), U.S. Army Construction Engineering Research Laboratories (USACERL). The USACERL principal investigator was Edward J. Japel. Dr. Spencer Rugaber, Hernan Astudillo, and Terry Kane are associated with the Georgia Institute of Technology College of Computing, Open Systems Laboratory. Alan Moore is Chief, CECER-FF and Dave Joncich is Acting Chief, CECER-FL. The USACERL technical editor was William J. Wolfe, Information Management Office.

LTC David J. Rehbein is Commander and Acting Director, USACERL. Dr. Michael J. O'Connor is Technical Director.



Contents

List of Tables and Figures

Tables

Figures

1 Introduction

Background

Many Army personnel can be classified as *knowledge workers*—people who produce not tangible products, but some form of processed or enhanced information. While most Army knowledge workers depend on computer processing to complete their tasks efficiently, their tasks are often complicated by the many computer platforms and software packages used to contain and convey needed information.

The U.S. Army Construction Engineering Research Laboratories (USACERL) has been conducting ongoing research into resolving problems of information access and management for knowledge workers, with the ultimate goal of developing a comprehensive performance support environment for this group. The Knowledge Worker System (KWS) is a prototype scheduling program designed to help knowledge workers organize and coordinate their work by storing task scheduling information in a centralized data base. KWS tracks scheduled events, lists completed events, and outlines the steps necessary to complete forthcoming tasks.

It may be possible to broaden KWS's applicability by converting it to an "open systems" technology, an approach that would make the program compatible with a number of different platforms. Open systems technology has become increasingly important in computing environments in recent years. Following open systems standards can reduce the overwhelming cost of software development, improve system reliability, and reduce maintenance costs (Quarterman and Wilhelm 1993). Software applications can be tested for adherence to standards; therefore, metrics can be developed to determine the portability and interoperability of applications. This study analyzed the feasibility of re-engineering KWS for open systems and outlined strategies to implement the conversion.

Objectives

The objectives of this study were: (1) to determine the feasibility of converting KWS to open systems technology, including POSIX and Motif, (2) to explore the feasibility

of reimplementing KWS in the DOD-standard computer language, Ada, and (3) to investigate the current market availability of language tools, POSIX-compliant operating systems, and Graphical User Interface (GUI) builder tools.

Approach

A statistical analysis of the KWS source code showed that developing the user interface would be the largest effort in re-engineering the program. On this basis, the study concentrated on locating Ada and C/C++-based user interface tools and support systems for Motif. A market survey was done to locate other open systems tools and operating systems. Information from the source code analysis and market survey was used to perform a transition study that concentrated on issues involving hardware platforms, operating systems, programming languages, user interfaces, and organizational decomposition (object-oriented vs. functional programming characteristics).

Scope

This study analyzed Knowledge Worker, Version, 1.6. Results of this study may not apply to later versions of the program.

2 System Overview and Conversion

The Knowledge Worker System

The Knowledge Worker System (KWS) is a software package designed to simplify the job of knowledge workers (Construction Research Center 1993). KWS allows knowledge workers to organize and coordinate their work by storing task-scheduling information in a centralized database. KWS tracks the scheduled events and any modifications to the schedule, and also serves as a repository of information about each task. KWS helps keep knowledge workers on schedule by providing a list of tasks to be completed and outlining the steps necessary to complete each task. It notifies the user of schedule or task changes and retains completion data for supervisors.

Open Systems Organizations, Standards, and Conventions

Identifying the major standards organizations and their activities is key to understanding the open systems world. This is a rapidly developing market, and therefore it is crucial to continually monitor the journals and newsletters from the various organizations. The following describes important standards organizations, relevant standards, and some definitions of conformance.

Open Systems Organizations

This section describes the various open systems organizations, their current status and purposes, and the relationships between them.

Uniforum. Uniforum is a nonprofit international association of open systems professionals that publishes the *Uniforum Monthly*, a journal of open systems and UNIX articles, and *Uninews*, a biweekly newsletter. Uniforum also publishes the annual *Uniforum Products* directory to promote trade and communications within the community. It also publishes a series of technical guides and overviews for open systems topics.

X/Open. Established in 1984, X/Open is an independent international consortium of computer system vendors who share the goal of developing a common applications

environment for multiple vendors based on international and *de facto* standards. Most of the largest industry vendors and customers are members of this consortium. X/Open is developing the Common Applications Environment (CAE), which contains practical interface specifications for interoperability and software portability. X/Open is more concerned with practicality than with formality, and has adopted and adapted existing standards as a basis for the CAE. The CAE is being developed through three programs:

1. *The Xtra Market Requirements Process*: This process identifies the real market needs for applications in open systems environments. The results of this analysis give X/Open a consensus view of the market requirements. The Xtra process also creates and guides technical work groups for specific issues.

2. *The XPG Specifications*: The X/Open Portability Guide (XPG) is a set of specifications that define an open systems environment interface. The XPG includes an integrated set of components needed by a portable application.

3. *The X/Open Conformance Testing and Branding Program*: X/Open publishes the *X/Open Portability Guide*, which contains an extensive set of conformance criteria based on verification tests. The VSX3 test suite exists to verify that the system software running on a hardware environment conforms to the X/Open specifications. The test suite produces a report that rates the product's X/Open conformance. Products deemed compliant receive the X/Open "brand," which symbolizes its acceptance.

IEEE 1003 Committee. The 1003 series of committees were chartered by the IEEE society to develop the standards documents for the Portable Operating System Interface for Computer Environments (POSIX). These all-volunteer committees represent a cross-section of expertise from industry and academia. IEEE standards are subject to reaffirmation every 5 years, which means that the POSIX.1 standard will be due for review in 1993.

ISO. The International Standards Organization has been involved as a review body in the development of the POSIX.1 standard (approved as a Draft Proposed International Standard). Some minor changes were suggested so the POSIX.1 standard could be submitted as a full international standard.

NIST. The National Institute of Standards and Technology originally developed its own operating systems standards, but has since merged with the IEEE 1003 committee to develop POSIXFIPS. This standard mandates some features considered optional or unspecified in POSIX.1, but otherwise matches the POSIX

standard. NIST also produces the Application Portability Profile, which outlines a set of standards for application development.

ANSI. The American National Standards Institute has not been involved in the development of the operating systems standards, but it has been involved with the development of C language standards (ANSI C) that include standard libraries and operating system interfaces. ANSI is working with the POSIX.1 committee to address these OS-specific functions.

POSIX Conformance

The major goal of standardization is to provide a platform for portability and interoperability. This is accomplished through a variety of mechanisms with varying degrees of formality. Conformance to the standards also ranges from formal certification to partial compliance. The following discussion outlines how the standards differ and how conformance is measured.

Definitions. To assess compliance, the specification's formality must be determined precisely. Table 1 lists the terms defined for their use in this study.

IEEE 1003. The POSIX operating system specification is a formal standard, IEEE P1003 and ISO/IEC IS 9945. The formal standard is part of a larger body of work that includes many projects and draft standards, some of which are in balloting. Table 2 shows the relevant IEEE specifications and standards.

Table 1. Definition of terms related to standards compliance.

Term	Definition
Standard	A formal specification reviewed and approved by a formal standards body such as ANSI or NIST.
Specification	Not necessarily a standard, but may be in the review process to become a standard.
de facto standard	A specification that is not a formal standard approved by a standards organization, but so widely used that it is recognized as a standard.
Profile	Defines an application interface or environment with a set of specifications and standards. Profiles may be standards produced by an open systems organization, or may be specific to a vendor.
ISP	An internationally standardized profile.

Table 2. IEEE POSIX standards and specifications.

IEEE Spec	Scope
1003.1	POSIX System Application Programming Interface (API)
1003.1a	Extensions to 1003.1
1003.2	POSIX Shell and Utilities
1003.2a	User Portability Extensions (UPE)
1003.3	POSIX Test Methods Standard
1003.4	Real time extensions (including threads)
1003.5	Ada bindings to 1003.1

The Test Methods Standards committee (1003.3) has two subcommittees: 1003.3.1, which is developing test methods for 1003.1 (System API), and 1003.3.2, which is developing similar methods for 1003.2 (Shell and Utilities). Other POSIX committees are charged with developing their own test methods.

Testing for compliance is performed by laboratories that have been accredited by authorized accreditation bodies (such as NIST). Then an independent validation body validates the results of the tests. Finally, the accredited laboratory provides certification for the tested products. Conformance to the above-listed standards and specifications can take two different forms: application and implementation of the system interface.

Application Conformance. Conformance to the POSIX.1 standards for applications determine the level of portability of that implementation. There are three levels of conformance for applications:

1. Strictly conforming. The application exclusively uses features from the POSIX.1 standard or applicable language standard.

2. Conforming POSIX.1. Conforms to the POSIX.1 standard, but may also use other standards not related to the System Interface Standard. All standards used must be documented, along with options and dependencies.

3. Conforming with Extensions. Conforms to the POSIX.1 standard, but may use nonstandard extensions or facilities. Implementation-defined behavior is acceptable but must be specified in the implementation.

Implementation Conformance. For system interfaces, there is only one form of conformance: the standard facilities of POSIX.1 must be implemented with the specified behavior. The concept of a "strictly conforming implementation" does not exist; implementations may support extensions, language bindings, and parameters,

as long as the basic facilities of the POSIX.1 standard are not altered and a strictly conforming application will perform correctly.

In fact, it is nearly unavoidable that the POSIX.1 standard be augmented in an implementation because the standard does not address such key features as system administration and some file-system support mechanisms. Therefore, vendors of POSIX-compliant systems must document the extensions and implementation-defined features of their interface.

Re-Engineering Methods and Techniques

An overview of current research in reverse engineering and re-engineering systems follows. Table 3 lists definitions of terms used in this document that relate to re-engineering methods.

Migration Strategies

There are many ways to move an information system to a distributed open systems environment. The following section lists some of the studied approaches organized roughly in order of increasing required effort.

Every strategy has its costs and benefits. For any given situation, the costs and benefits of candidate strategies must be compared to select the best approach. **Decision Criteria** (p 15) describes questions that can be asked to help clarify the situation before choosing a strategy. A description of the strategies themselves follows.

Table 3. Definition of terms related to re-engineering methods.

Term	Definition
Migration (or conversion)	A general term that refers to the procedures, methods, and practice of moving software from one computing environment (including hardware platform, operating system, and tool support) to a different environment.
Re-engineering	The task of redesigning and reimplementing code. Re-engineering may include changing an application's functionality as well as its implementation.
Porting (or transporting)	Moving an application from one environment to another with minimal changes. Porting usually implies that nothing other than machine-specific code is changed; the code, appearance, and functionality of the ported software should be nearly identical to the original.
Reverse engineering	The process of examining code from an existing application to understand its design.
Forward engineering	The process of reimplementing a system from a re-engineered design.

Doing Nothing. The baseline against which the other strategies must be measured is the strategy of simply doing nothing. In this case, there is no real benefit, and the cost is fairly well understood. This strategy may be appropriate if an application is going to be replaced or phased out, or if an application is used only infrequently at a single site. In such circumstances, there is little value in supporting open systems or distributed access.

Direct Application Porting. Sometimes a system can be re-engineered simply by directly porting the application to the new platform without adding any new functionality. To pursue this strategy, the machine-specific code is rewritten fc new platform, and then recompiled on the new platform. This is slow but rela straightforward. Once a list of specific conversions has been made, the source-code conversion can be partially automated.

The direct porting strategy is best when a large portion of the code is platform-independent. Simple porting is not possible if large portions of the code must be rewritten (for example, when replacing a user interface with very different display technology). Porting may also be used as an interim step to some of the strategies described below.

Conversion by Re-Engineering. Without a clear understanding of the original code's design, moving a system from one platform to another is an open invitation to disaster. Reverse engineering software reveals an application's structure; the understanding thus gained can then be used as a basis for enhancements.

The benefits of such an approach are obvious, but the costs are hard to measure. Reverse engineering is slow, hard work. Some mechanisms for partially automating the process are described below.

Manual re-engineering is best when the existing code will be used extensively for the foreseeable future. Maintenance activities that require modification of existing code (as opposed to adding new modules) can also help justify the expense of reverse engineering.

It is not always necessary to reverse-engineer an entire system, although the reverse engineer always needs to understand the relationship of the part under inspection to the whole system. Thus, when documentation is plentiful and accurate and the maintenance personnel are experienced, partial reverse engineering may be more feasible.

Automatic Reverse-Engineering. Automation can help reduce the cost of reverse engineering. Automatic reverse engineering involves using a program that identifies features of existing programs and translates them into a standard design representation. Unfortunately, few tools currently exist for that task, and those that do exist are primitive, capable of describing only the existing system's surface features. The problem of analyzing code is compounded by the fact that layers of bug fixes and rewrites pile up on the original code like patches on old clothes, so that the original design is often obscured.

Programs are analyzed by systematically inspecting the source code itself; the fruit of the analysis is a description of the application domain and of the procedures that the program models. This analysis can be performed manually, but the process is labor-intensive and slow. Any help an automated tool offers in this area is a blessing.

Some CASE tools, such as IDE's Software Through Pictures (STP), support reverse engineers in a variety of ways. The information in a diagram is stored in a standardized text file with a well-documented format. Diagrams are normally constructed in STP by the user, who manually selects icons to flesh out the diagram on the screen. Using the published file format, however, diagrams can be constructed automatically based on the information extracted by other tools. This representation can help to forward-engineer to a new platform.

Rewrite From Scratch. A final strategy must be mentioned for reasons of completeness. Sometimes it may be best to throw out the existing program entirely, respecify the requirements, and then rebuild the whole system from scratch. This decision is most appropriate when an old system needs to be greatly modified and is so complex that re-engineering would be more expensive than simply starting over.

Decision Criteria

The following section describes the variety of costs and benefits examined in selecting a strategy, or the "decision criteria."

Factors Related to Usage of the Existing System.

- How many users does the system have?
- How are the users distributed topologically? (Are they logged into the mainframe, do they submit batch jobs, or are run requests handled manually?)
- How frequently is the application used?

- In what ways is the application used? (What is the ratio of data updates to reports produced? How frequently is each such use made?)

- What is the physical process by which the application is currently used (data entry, validation handled separately; manual or electronic distribution of reports)?

- How many different sites use the existing system?

- What is the expected lifetime of the existing application? Is usage increasing or decreasing?

- In terms of human and machine resources, how much does it cost to execute the program? How does this cost vary across the various types of uses?

- Are there political factors that would impede the reduction in information control that results from distributed access?

- Are there administrative procedures that would be difficult to provide in a distributed environment? What are the costs in transforming these procedures?

- Do other applications depend directly on the data produced by this application? Conversely, does this application depend on the products of other applications?

Factors Related to the Structure and Functionality of the Existing System.

- How compatible is the current architecture with the client/server model? Is the application primarily batch or interactive?

- What external resources and connections does the application require? How extensively are these used?

- Does the existing system make use of nonportable operating system capabilities? Does the existing system interface to other existing systems?

- Does the existing system write reports? If so, is the computational functionality separable from the report construction functionality? Are there reports that could be replaced by Structured Query Language (SQL) queries? Are there reports that could be replaced by reports constructed by the relational data base management system (RDBMS) report writer capability?

- Does the current application do significant data validation that could be replaced by the data validation features of the RDBMS? Could the current application make effective use of advanced RDBMS operations like views and joins?

Factors Related to Expected Usage of the Converted System.

- How much more frequently will the system be used when it becomes available on a network? What is the expected change in the kind of usage (e.g., from batch to interactive) promoted by distributed access?

- Can the application take advantage of DBMS capabilities such as security and integrity?

- What is the expected change in execution cost in terms of machine and human resources?

Factors Related to Expected Evolution of the Converted System.

- Does the existing system make use of a DBMS? Is it relational? Does the existing system make use of an older COBOL version? Are there portability issues related to data conversion? Can this application be integrated into others?

- How much effort is now put into maintaining the system? What enhancements to the system are planned? What enhancements would be facilitated by the use of an SQL interface to the data?

- Are there personnel available who are familiar with the internals of the existing system? Is the system documented? How up-to-date and accurate is the documentation? Is the money available for a comprehensive reverse engineering effort? Does this include funding to support the training of users in 4GLs? Is incremental conversion feasible?

- Is the application part of the effort to standardize the use of data item names? How closely does it conform to these standards?

3 Vendor Survey

An important factor in the feasibility of re-engineering Knowledge Worker using open systems technology is the availability of tools and resources. A description of the survey and evaluation of open systems and supporting products on the current market follows.

Operating Systems

For this study, a primary concern with operating system software is the level of POSIX compliance. UNIX systems for the Sun SPARC architecture and also for the 386 personal computer (PC) architecture were examined.

SunOS 4.1.x System V Environment (Platform: SPARC)

The SunOS version 4.1 installed with the System V installation option is certified POSIX-compliant. It is actually a superset of the POSIX.1 standard, including all of the functionality of the standard plus additional SunOS functionality. Working in the POSIX environment under 4.1 simply entails adding the POSIX libraries to the user's path.

Sun Solaris 2.0 (Platform: SPARC and 386/486)

Like SunOS 4.1, the latest release of the Solaris operating system is POSIX-compliant. Solaris 2.0 is not binary compatible with SunOS 4.1.x, however, so application tools to check for implementation on Solaris 2.0 must be chosen with care. It was recently announced that Solaris for PCs will be available mid-July 1993.

Microsoft Windows NT (Microsoft) (Platform: 386/486 and SPARC)

Microsoft's most recently announced operating system is partially POSIX-compliant. It implements the base functions of POSIX 1003.1 but is not complete. The POSIX compliance is provided in a subsystem that is not Windows-compliant. Windows applications are not POSIX-compliant. Recently, Microsoft announced Windows NT would be available on the SPARC platform.

Santa Cruz Operation (SCO) UNIX (Platform: 386/486)

SCO UNIX is a certified POSIX-compliant UNIX for the PC platform. It is a 32-bit, multithreaded, multitasking, multiuser kernel with virtual memory.

Ada Compilers

Since C compilers generally are provided with UNIX implementations, and the portable GNU C and C++ compilers are freely distributable, this survey concentrated on Ada compilers.

Verdix 6.0 (Verdix) (Price varies by platform)

The Verdix Ada Development System (VADS) is an integrated set of software tools for Ada program development. The package includes a validated Ada compiler, an interactive debugger, a library management system, and other tools. VADS is available on a number of platforms, including Sun SPARC, HP, DEC, and IBM PC (under AIX). The VADS system is partially POSIX-compliant, and is being staged to be fully compliant. The next release is due in August 1993 and will support IEEE 1003.1 Chapters 2, 4, 5, and 6. The following release is scheduled for December 1993 and will add some low-level features, including Ada I/O and signals.

SPARCWorks Ada (SunPro) (List $10,000)

SPARCWorks Ada is a "value-added" version of Verdix 6.0 for the Sun SPARC platform. As such, it has all of the features and capabilities mentioned above, plus integration with Sun display tools, such as *devguide* (a GUI builder for Open Look that eventually will be rewritten to handle Motif). SPARCWorks Ada can be purchased with a maintenance option that will include the POSIX upgrades this summer and next winter.

Alsys Ada (Alsys) (List $7,500)

Alsys Ada is supported on many platforms, including SPARC, SCO UNIX, and HP. The vendor claims that it is POSIX-compliant and is capable of producing POSIX-compliant code. Alsys Ada is a complete development environment including compiler, library manager, and symbolic debugger. The AdaProbe symbolic debugger and the AdaXref cross-reference generator are included, along with the AdaMake makefile utility. Alsys also provides access to Motif through the "Ada Tune" tool ($2,250) and to the Xlib and Motif libraries ($2,995).

Ada Native and Cross Compiler Systems (TLD Systems) (List $10,000-$80,000)

TLD provides a POSIX-compliant Ada development system with cross-compiling capabilities for real-time embedded systems development.

Graphical User Interface Tools

Building a Graphical User Interface (GUI) can be made much easier with GUI builder tools. Some of the toolsets listed here are libraries or widget (generic tool) sets, and some are actually palette-based tools that allow the user interface to be built in "drag-and-drop" fashion. These GUI builder tools then generate the X and Motif code to produce the user interface in the application. Because Sun announced that Open Look is being discontinued in favor of Motif, only Motif-based tools were investigated.

Motif Toolkits—C and C++

Motif toolkits use the underlying native toolkits and provide widgets, gadgets, and palette-based GUI builders. These tools produce C and C++ code to generate the interfaces.

UIM/X (Visual Edge—Bluestone, distributor). (List $5,000.) Reputedly the best GUI builder on the market for Motif, UIM/X includes a native toolkit and an interactive GUI builder. UIM/X also includes an interpreter that allows developers to test interfaces without going through the time-consuming steps of compiling, linking, and debugging the code. Researchers at Georgia Institute of Technology received and installed a demo copy of UIM/X and reported that it was a very powerful program.

Builder Xcessory (Integrated Computer Solutions, Inc.). BX is a tool for building Motif GUIs with a C interface. It also includes a "drag-and-drop" capability for style sheets. The Army Research Laboratory (ARL) at Georgia Institute of Technology has used BX for Motif development and warns that BX's own user interface is cumbersome and produces a nonstandard user interface.

Centerline Software (ViewCenter). (List $2,995 + $995 for libraries.) This SPARC-based GUI development tool supports Open Look and Motif. It is basically a GUI builder with hooks to C++. It implements its own toolkit (as opposed to using a "native" toolkit) to lend a particular "look and feel" to the developed applications.

C++ Views (Liant Software). (List $1,495 [UNIX]/$494 [Windows].) The Views package supports Motif and OS/2 Presentation Manager with the native toolkits. It includes an application programming interface (API) but no GUI builder tool.

Objectbuilder (ParcPlace Systems). (List $2,995.) Objectbuilder is a C++ programming tool that supports OpenLook and Motif for the SPARC platform only. It is a GUI builder but does not have its own native toolkit.

Motif Toolkits—Ada

These toolkits are similar to the Motif toolkits above, except that they generate Ada code instead of C or C++.

UIL/Ada and Ada/Motif (SERC). (One copy, $2,995; less for multiple copies.) This tool translates the output of palette-based GUI builders (such as UIM/X) into Ada, allowing Ada applications to be built with rapid prototyping. The UIL/Ada tool translates the intermediate representation from the palette builder and produces Ada code with Motif binding calls. The Ada/Motif libraries support calls from Ada to Motif. These tools work with the Sun Ada compiler (which is not POSIX-compliant), but not with the SPARCWorks Ada compilers specifically. It also works with SCO/Alsys Ada and HP/Alsys Ada.

GRAMMI (EVB Software). (One copy, $5,000; 2-5 copies, $4,500 each.) GRAMMI is an Ada user interface toolkit that supports the development of GUIs using the X windowing system. The GRAMMI widget set is written in Ada and is based on, but not completely compliant with, the Motif look and feel. The User Interface Editor allows palette-style rapid prototyping. GRAMMI works with SunAda and HP/Alsys Ada.

STARS Repository Motif/Ada Bindings. (Free [public domain].) The STARS (PAL, formerly SIMTEL-20) repository is a collection of public domain software that can be downloaded from the Internet. Among these is a set of bindings developed by Boeing that consists of a library of Motif widgets callable from Ada programs. This is *not* a GUI builder tool, but simply a library. Researchers at the Georgia Institute of Technology are investigating the pathnames to obtain these files and will download them. These bindings should work with all or most Ada compilers.

Portable GUI Development Toolkits

Several tools currently on the market are advertised as GUI development tools for portable applications. This means that the designer can write code for any one API and then link to libraries that govern the look and feel of the application on each

different platform. This option looks very attractive at first, since theoretically they could enable the development team to write and maintain just one version of the source code for KWS, which would compile correctly for both Windows and Motif. However, in practice the tools have shortcomings. The following paragraphs give the results:

XVT Portability Toolkit (XVT Software). (List $1,450-$4,400.) The XVT toolkit is advertised to support GUI development for Microsoft Windows, Macintosh, Motif, Open Look, and character interfaces, among others. It includes a native toolkit and a GUI builder (a WYSIWYG "palette" tool).

Developers at the Georgia Institute of Technology who have used this tool to develop an application that had both Microsoft Windows and Motif user interfaces strongly recommended against using it. They said that the resulting interfaces were nonstandard and did not conform to the "look and feel" of either Windows or Motif. They also stated that even though XVT's advertisements claim that programmers only need to use a single API, it was necessary to go into the generated code to customize and fix problems, which slowed development. They are so frustrated that they are considering taking a loss on their sizable investment in XVT and starting over from scratch, developing two separate interfaces using Windows-specific and Motif-specific toolsets (and having two copies of the source).

Aspect (Open Inc.). (List $3,995.) Supports Microsoft Windows, Macintosh, Motif, and Open Look. Aspect includes a native toolkit and a GUI builder, similar to XVT.

Open Interface (Neuron Data). (List $7,000 [$15,000 for developers].) Supports Motif, Open Look, Windows, Presentation Manager, Macintosh, and character interfaces. Neuron Data uses its own proprietary toolkits to achieve the Windows and Motif look and feel (rather than the native, standard toolkit such as XVT uses). Neuron feels that this approach enables the company to produce a more flexible product than it could if it stuck with the native toolkits. Because of the proprietary implementations of the toolkits, it is likely that this tool diverges from the standards.

Database Access

KWS currently relies on a centralized Oracle server. Since SQL is a standard 4GL, KWS could be generalized for other database server programs. However, it is assumed here that Oracle will remain as the server. The following tools are provided for application programs to interface with Oracle servers.

Pro-Ada

Pro-Ada provides an application programming interface to an Oracle server, callable from Ada. Interfaces are provided by the SPARCWorks Ada and Alsys Ada compilers.

Pro-C

The corresponding application programming interface to the Oracle server, callable from C programs. Modules written in Pro-C can be linked with modules from other C compilers.

4 Transition Study Results

This chapter describes the transition study, which examined platform, operating system, user interface, language, and organizational decomposition issues.

Statistical Analysis

To gain insight into the nature of the KWS application, a statistical analysis of the source code was done. This allowed an assessment of the relative importance of these issues according to the amount of code devoted to each of the areas of study, and a determination of the areas that would most affect the re-engineering effort. The statistical information was gleaned through a combination of techniques:

1. *Inspection and Analysis.* The most tedious and labor-intensive way to learn how code functions is simply by reading code and comments. This method is used to make subjective judgments, e.g., on code and comment quality.

2. *Developer interviewing.* The re-engineering process is made considerably easier if the original developers of the candidate system are available for interviewing. KWS system developers were interviewed to get their estimates of complexity and to help with difficult areas.

3. *Automated tools.* Automated tools can quickly and efficiently answer statistical questions that could take hours if calculated by hand. The UNIX tools *grep* (which searches text files for patterns), *wc* (which counts words) and *diff* (which compares files) were used extensively to examine the source code for occurrences of system calls, interfaces to databases, and other statistics.

Statistical Analysis Results

Table 4 shows the initial analysis of the code resulting from the automated tool method. The number of lines of code (LOC) indicates KWS is a medium-sized application. KWS depends on two interfaces: the Microsoft Windows Application Programming Interface, which implements the GUI, and Oracle, the database interface. These areas were examined to determine how much of the code is

platform-specific and therefore will need to be rewritten. Table 4 shows the initial analysis of the code, done mostly with automated tools.

Table 4. Statistical analysis of KWS source code.

Category	Total
Total lines of code (LOC) for KWS	38,600
Total lines of executable code (LEC)	29,600
Total number of source files	97
Number of executable modules	39
Number of header files	44
Misc. files (defs, etc.)	14

The next step was to examine the code to determine percentages that might give us information on the level of difficulty for migration. Table 5 shows the distribution analysis of the code.

Conclusions From Statistical Analysis

This analysis reveals that the overwhelming majority of the code is in the user interface. Therefore, the largest part of the re-engineering effort will center on rewriting the Microsoft Windows-based graphical user interface to conform to X and Motif functionality. Due to the differences between Microsoft Windows and Motif, this will probably entail some redesign as well as re-engineering.

The next most significant piece of the Knowledge Worker code is the database (Oracle) access code. This code may be easier to re-engineer than the user interface because the actual SQL calls will probably remain the same. Therefore, the re-engineering task will probably entail mostly syntactic changes, but the basic structure and flow will not change.

The system-dependent file I/O and process interface code will need to be re-engineered because of the substantial differences between the Microsoft Windows operating system and the UNIX/POSIX operating system. The remaining algorithmic code, just 2 percent of the whole, is the only code that could probably be used unchanged after a re-engineering to open systems.

In summary, the large majority of KWS code is platform-dependent and therefore will have to be re-engineered for the open systems environment.

Table 5. Distribution analysis of KWS source code.

Code	Module	Amount
User interface code		85%
Algorithmic code	Scheduling module	2%
System interface code	Database access	9%
	File I/O	3%
	Process interface	1%

Hardware Platform Issues

The largest issue in platform dependence is the availability of tool support and the differences between operating systems. A review of the Knowledge Worker code showed no hardware dependencies not handled by the operating system. Therefore, no platform-dependent problems that are not already addressed by the operating system conversion are anticipated.

Because the Sun SPARC platform is the most common UNIX workstation, and because the most comprehensive set of development tools exists for this platform, the initial re-engineering to POSIX and Motif will be done on the SPARC. A later phase of the project will include a true port to a totally different common architecture, the 386 /486 PC-based UNIX.

Operating System Issues

This section details issues that arise in re-engineering from Microsoft Windows to the UNIX/POSIX environment. Part of this study entailed an attempt to devise mappings from Windows capabilities to POSIX features. According to the statistical analysis of the code, approximately 5 percent of the code is OS-dependent. Other than the services mapping described below, the only issues are differences in the file systems. UNIX file names are case-sensitive, while Microsoft Windows filenames are not. There are also syntactic differences in the filenames that must be taken into account.

Operating System Services Mapping

To assess the feasibility of supporting all of KWS's functionality in an open systems technology, the number of Microsoft Windows operating system calls were examined (Rector 1992) and an attempt was made to map these calls to the corresponding POSIX system calls defined in IEEE 1003.1 (IEEE 1988). All of the OS-specific calls could be mapped to POSIX calls, so all of KWS's functionality can be supported with open systems. Following is the mapping of KWS Microsoft Windows operating systems services to the POSIX calls that fulfill the same functions.

File Manipulation (open, fopen). The Windows Open and Fopen calls are supported in POSIX as specified by IEEE 1003.1 in section 8, referencing the C Language Standard. Therefore this functionality is present and can be translated.

Global Memory Allocation (GlobalAlloc). Dynamic memory allocation in Windows is handled with the GlobalAlloc system call. Memory blocks may be fixed or

moveable. The POSIX.1 standard specifies that dynamic shared memory allocation must conform to the C Language standard for the C library call *malloc*. In Ada, dynamic memory allocation is performed in the language itself instead of with a direct system call, via the "new" operator on an access variable. Global dynamic memory allocation, therefore, will not be a problem with either C or Ada.

Task Creation (child windows). In a KWS application on Microsoft Windows, child task creation is actually a function of the user interface. Here this will be handled with the Motif XmCreate() calls. (There are 57 different calls, depending on the type of child widget or gadget desired.) Therefore the "proof of concept" included an experiment with mappings from Microsoft Windows child window types to Motif widgets.

Note that the implementation of KWS does *not* use some of the features of Windows that are not supported directly under POSIX, such as Dynamic Data Exchange (DDE), Dynamic Link Libraries (DLL), and process communication (SendMessage).

Language Issues

The current implementation of KWS for Microsoft Windows is written in C. However, it has been shown that only 2 percent of the code (the algorithmic scheduling module) could potentially be ported directly. Since the great majority of the code must be re-engineered anyway, the language issues then revolve mostly around tool availability and support. This section contrasts the advantages and disadvantages of the two candidate languages, C and Ada.

Standardization

The DOD standard 1815a defines the Ada language, which is now also an ANSI standard. The Ada language may not be subsetted or supersetted if the compiler is validated. There is also an ANSI standard for C. If ANSI C is adhered to, then C is fairly portable. In addition, there is a standard UNIX tool, *lint*, that can evaluate how closely source code conforms to ANSI C.

Compiler Availability

Ada compilers are now available for almost every hardware platform, though they tend to be more expensive than C compilers. Some POSIX-compliant compilers are available, and more are scheduled to come on the market soon. C compilers and libraries are usually provided with UNIX distributions, and high-quality public

domain C and C++ compilers (the GNU toolset) are available free of charge.* C programs can use the POSIX libraries of any POSIX-compliant UNIX implementation without modification, since the POSIX interface was originally specified for C.

Graphical User Interface Tool Support (GUI builders)

There are GUI builders available for both C and Ada, with slightly more tools available for C. Some of the tools produced C code, which then could be turned into Ada through a translation step. Public domain Motif bindings are available for both Ada and C.

Portability

Ada is designed to be portable and to support good software engineering practices such as information hiding, encapsulation, modularity, and fault tolerance. If compiler-dependent features are avoided, then code written in Ada is very portable. C is also portable, and compilers for the language are ubiquitous. However, C has many more possibilities for divergence than Ada. If C is chosen, the ANSI standard C should be adhered to for maximum portability.

POSIX Compliance

The POSIX specification was originally defined for the C language, so those bindings already exist. Recently, IEEE 1003.5, Ada language bindings to POSIX, were approved. Market vendors have responded that several POSIX-compliant Ada compilers will be available by summer 1993.

User Interface Issues

A recent major announcement from the six major UNIX vendors (Sun, Hewlett-Packard, Univel, IBM, UNIX Systems Laboratories, and the Santa Cruz Operation) detailed an effort for these vendors to cooperate on developing a Common Open Software Environment (COSE, pronounced "cozy") (Uniforum Press Release 1993). This means that the desktop environment between all the vendors will be the same—and that desktop applications will be common across all the platforms. This does not mean that the underlying UNIX operating systems will be standardized, but that the user interface to the desktop will be standardized. This announcement

* Through the Center for Software Reuse Operations, 500 N. Washington St., Falls Church, VA 22046, tel. 703/536-7485.

confirms and strengthens the industry commitment to the concepts and standards of open systems.

One major effect of this announcement is that Sun has decided to drop development of its Open Look environment and toolkit. COSE will be based on SunSoft's ToolTalk services and the Motif toolkit with some compatibility enhancements, and some features borrowed from the technically superior Open Look. Existing applications using XView and OLIT will still be supported.

The effect on the Knowledge Worker migration is that it will now be re-engineered for Motif, by default. Since the statistical analysis showed that 85 percent of KWS code is devoted to the user interface, this is a primary area of concern. For this reason, the user interface was prototyped as a Proof of Concept (Chapter 5). Because of the differences between Microsoft Windows and Motif, the user interface will need to be re-engineered. Some small changes in the appearance of the user interface will also be necessary. These are detailed in the next chapter.

5 Proof of Concept

A GUI rapid prototype was built to demonstrate the feasibility of using open systems GUI tools to re-engineer KWS. Since 85 percent of the KWS source code is user interface code, the user interface is the most important component to re-engineer to assess the difficulty of the migration. The Appendix to this report contains graphical representations of the screens generated for the prototype.

Overview

The prototype of the KWS user interface is meant to show representative paradigms, differences, and problems in re-engineering from Microsoft Windows to UNIX and open systems. For expediency and availability, this prototype was built using the Sun tools Dev/Guide and the Open Look toolkit. The interface was translated item by item.

Transformations and Assumptions

The fundamental problem in translating one interface into another while preserving its functionality is the different stylistic conventions. For example, Open Look (OL) does not have menu bars as other toolkits do, and OL applications do not have "Quit" options, because this is handled by the window manager. The best that can be done is to make the functionality and "look and feel" as close as possible, while respecting the conventions of both platforms.

The main concerns have been, first, the item's functionality, and, second, its appearance. For example, a Windows menu title that drops down a menu when clicked must be mapped to a OL button that shows a menu when pressed and has a similar label, shape, color and position. The label and position of an item can be inferred easily from the manual's figures and from actual KWS use. The color (where applicable) can also be inferred from use. But the position and the way items are grouped sometimes does not directly correspond because of different button sizes or alignments.

Mapping the Interface

Table 6 characterizes the mapping that was used for the prototype. The three steps in translating each item are:

1. Determine the equivalent Open Look item to correspond to the Microsoft Windows item.

2. Customize Open Look item for similar behavior (e.g., show menu or display user list).

3. Customize Open Look item for similar look (e.g., label and position).

Table 6. Windows-to-Dev/Guide mapping.

Windows Item	Dev/Guide Item
[a1] menu bar	[a2] rectangular control area
[b1] menu button in (a)	[b2] button in (a) • set Type to "abbreviated menu" • set *Menu to proper menu*
[c1] menu	[c2] menu • set "not pinnable" • set Label to c1's label
[h1] menu option • not selectable	[h2] menu item • set "Inactive"
[d1] sub-menu in (h1)	[d2] menu • set SubMenu in (h2) to menu
[e1] scroll area	[e2] scrolling list • set ReadOnly as required
[f1] line in scroll area (e1)	[f2] item in scrolling list (e2) • set Item Label to line contents
[g1] menu when (f1) pressed	[g2] set "SubMenu" in (e1)
Other: The Attachment window has been translated into a TextPane, which loads a file when opened; this corresponds to the exact behavior of KWS.	

6 Transition Plan

This chapter outlines the choices and alternatives available for devising a strategy to convert the Knowledge Worker System using open systems technology and to perform a validation step to assess its portability.

Platform Choice

As described in Chapter 4 (p 24), the Sun SPARC architecture is the recommended choice for the first re-engineering effort. This platform was chosen because of its ubiquity and because it supports the best set of development tools currently on the market. Once the re-engineering effort is complete, then the open systems version of Knowledge Worker can be ported to other POSIX-compliant architectures. The resulting system should be ported to another open systems platform, a 386/486 architecture running SCO UNIX. This will provide a validation step to ensure the portability of the application and to test the quality of the open systems interfaces.

Strategic Alternatives

Alternative 1—Nonspecific Graphical Interface Tool

At first glance, the nonspecific graphical interface tool builders seem very attractive. In theory, the Knowledge Worker System could be re-engineered to the proprietary language of the tool, so that code could be automatically produced for each of the different user interface technologies. This would allow one version of the source code to be maintained that would produce code for Microsoft Windows, Motif, Open Look, and even Macintosh. However, after studying these tools further, and after hearing from large development projects that have used them, there proved to be serious flaws:

- The developer becomes locked into a proprietary intermediate language. This is dangerous for several reasons—the vendors have total control over the representation of the language and could change it at their discretion, causing major difficulties. Also, using a proprietary language violates the spirit of open systems.

- The tools are not robust or precise enough to completely specify the interfaces, requiring changes in the generated code to achieve the desired effects. This means the different platforms wind up with different versions of the source code, which is exactly what this technology is supposed to prevent.

- The SQL interface might differ on different hardware platforms, meaning once again that different platforms would have to have different source code.

- In general, the interfaces generated by these tools are inferior to hand-built interfaces. The builders are, unfortunately, not specialists in any single operating system. The interface technologies supported by these tools vary widely enough so that no one tool currently supports all of the systems well.

These tools are so primitive that they do not currently fulfill their promise. Therefore, their use is not currently recommended, although the technology will likely improve.

Alternative 2—Motif-Specific GUI Tool

The second alternative is to maintain two separate sources—the existing one for Microsoft Windows and the newly re-engineered open systems source using a Motif-Specific GUI builder tool. The palette-based tools now available are quite adequate and can significantly enhance the development process. Since tools already exist that can support Motif, POSIX compliance, and Ada, this is the recommended strategy for re-engineering the user interface.

Language Issues

Part of determining the feasibility of re-engineering KWS to open systems technology was to evaluate the tool support for Ada. There is already adequate tool support for Ada development. Since Ada is the standard DOD language, it is the recommended language, bearing in mind that:

- Tools are currently available, although the tool choices are rather limited and the tools are relatively expensive.

- Ada technology for open systems is still nascent, and some of the re-engineering work may need experimentation to solve emerging problems.

- Since the original KWS is written in C, the Ada implementation will totally diverge from the original; there will be no code sharing. In all future versions

of KWS, it will be necessary to modify both the original and the re-engineered Ada versions of the software.

- The expertise pool for developing and maintaining Ada applications is more limited than the expertise pool available for C.

The C language was considered as an alternative, and although Ada is the recommended choice because of the DOD standard, a re-engineering in C would also be a feasible, simpler task. C would offer some advantages:

- Some of the code (less than 5 percent) would not have to be re-engineered and could be used directly.

- Using C would reduce the amount of experimentation necessary, and therefore it would improve the accuracy of cost and schedule estimations.

- C tools are widely available and comparatively cheap or even free.

- The interfaces for open systems (notably POSIX and Motif) are defined in C, and therefore are the best-tested, the most reliable, and the most widely available.

While the considerations listed above are advantages, they are not strong enough advantages to advocate a waiver for C.

Strategy Recommendation

This chapter details the recommended strategy for re-engineering the Knowledge Worker System to open systems technology and Ada. Included are an overview of the development strategy and an associated manpower estimate, a development schedule, and a cost estimate for personnel services, equipment, and tools.

Development Strategy

The re-engineering effort should be performed in two phases. First there should be a re-engineering phase, choosing a development platform that has the best tool support for redesigning and reimplementing the KWS application. According to the vendor and tool survey done in this study, the Sun SPARC platform has the best development environment available for open systems tools.

The second phase of effort is a true port, moving KWS to another open systems platform to verify portability of the open systems design and code. Since the 386/486 PC is a ubiquitous platform, and since SCO UNIX is POSIX-compliant and is available for the 386/486 PC platform, it is recommended for the first migration. Subsequent migration platforms can be included as needed.

Schedule

Figures 1 and 2 show modified Gantt charts giving projected completion schedules for Phases I and II of KWS migration.

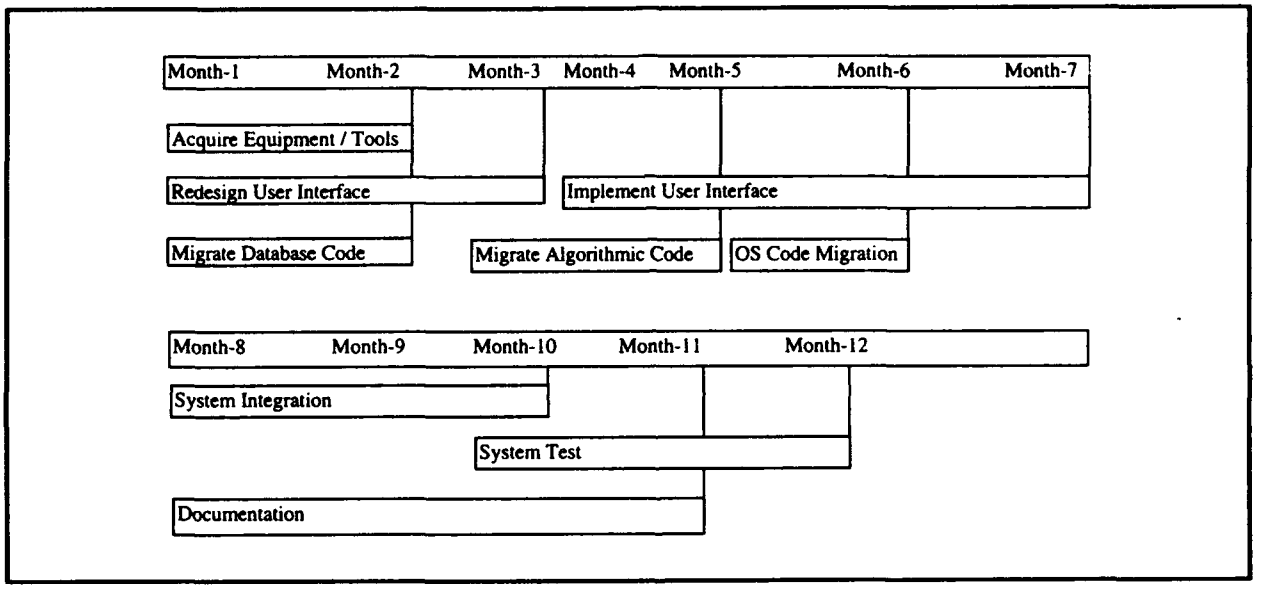


Figure 1. Projected completion schedules for Phases of KWS migration.

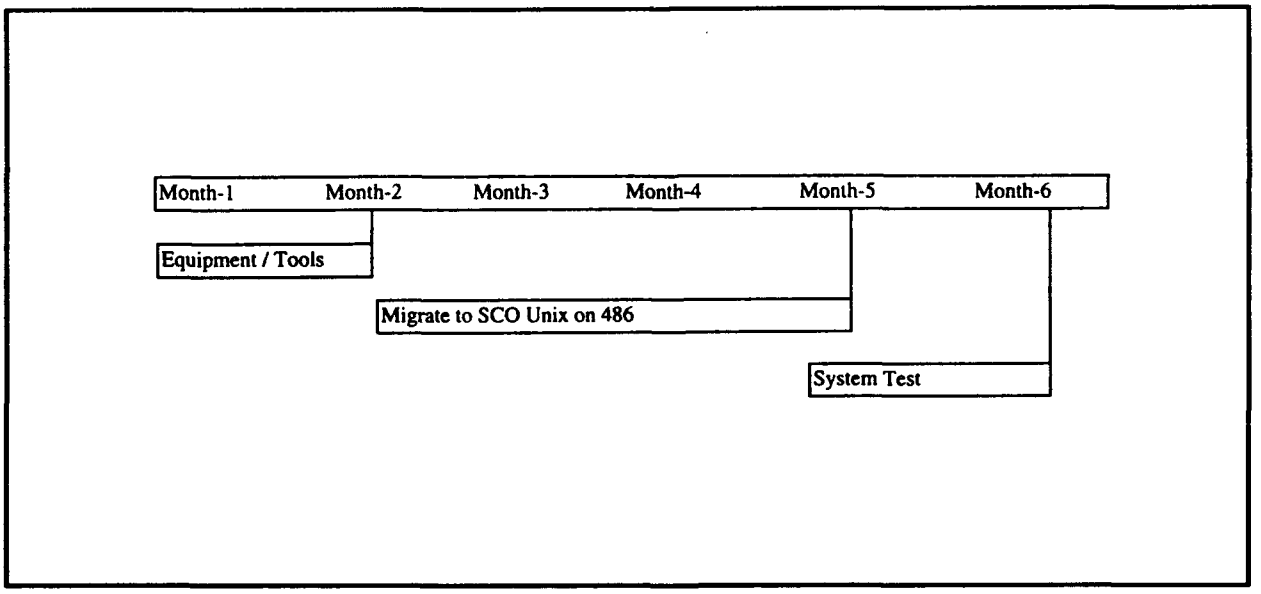


Figure 2. Projected completion schedule for Phase II of KWS migration.

7 Conclusions and Recommendations

This study concludes that it is feasible to convert the Knowledge Worker System to open system technology. An analysis of KWS showed that the program's source code is 85 percent user interface code, indicating that conversion of the user interface would make up the bulk of the conversion effort. A prototype graphical user interface (GUI) was built using open systems GUI tools to demonstrate the feasibility of accomplishing this major task. Such tools already exist to support POSIX, Motif, and Ada.

Since Ada is already the DOD standard, and because tools are currently available, Ada is the recommended choice for converting KWS to open systems technology. The re-engineering effort should take place in two phases:

1. A development platform should be chosen that has the best tool support for redesigning and reimplementing the KWS application. The vendor and tool survey done for this study indicated that the Sun SPARC station is best suited for this application, and is the recommended platform. The redesign and reimplementation of KWS should take place on this platform.

2. The Knowledge Worker System should be ported to a second open systems platform. Since the 386/486 PC is a ubiquitous platform, it is the recommended path for the first migration. Later platform migrations can be included as needed.

References

POSIX 1003.1 Specification (ANSI Standard) (Institute of Electrical and Electronics Engineers Inc, 1988).

IEEE 1003 Committee, *Technical Standards Reference Model*, International Standard 1003.3 (1988).

Human Computer Interface Style Guide, ADA 253475 38.92.

Knowledge Worker System Version 1.6 User Manual, Draft Automated Data Processing (ADP) Report (U.S. Army Construction Engineering Research Laboratories [USACERL], April 1993).

NIST, *Application Portability Profile, The U.S. Government's Open System Environment Profile OSE/1*, Version 1.0 (April 1991).

Quarterman, John, and Susanne Wilhelm, *UNIX, POSIX, and Open Systems*, Addison Wesley UNIX and Open Systems series (1993).

Rector, Brent E., *Developing Windows 3.1 Applications with Microsoft C/C++*, 2d ed. (Sams Publishing, 1992).

DOD *Architecture Implementation Concept for Information Systems: Technical Reference Manual*, Version 1.3 (January 1993).

Uniforum Press Release, *UNIX Leaders Announce Common Open Software Environment—Six Companies Agree on Software Technologies and Common Desktop, Reinforce Commitment to Open Systems* (San Francisco Uniforum Conference, 17 March 1993).

Appendix: Prototype Screens

This section contains some of the representative re-engineered open systems screens from the prototype. This shows the different look and feel in the open systems GUI. The prototype was developed using Sun's Dev/Guide palette-based GUI builder tool on a SPARC platform.

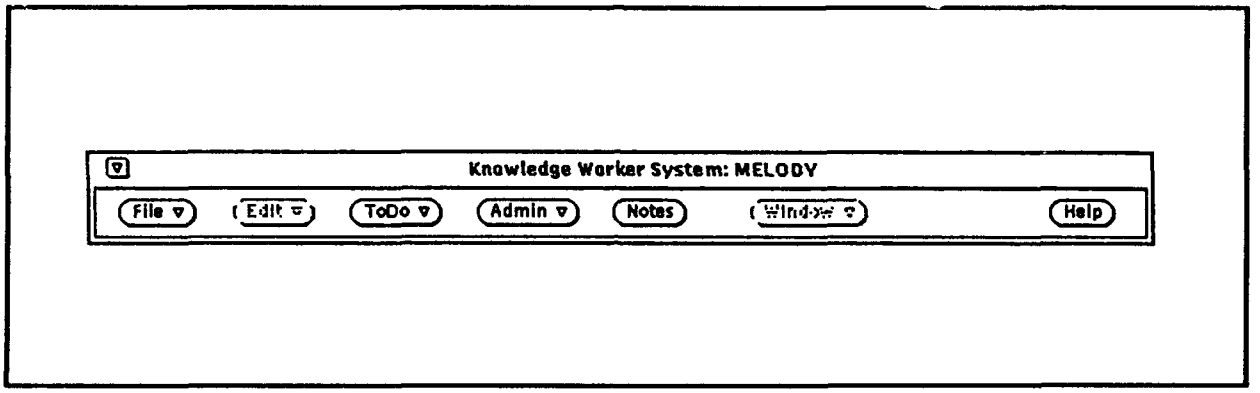


Figure A1. Main screen.

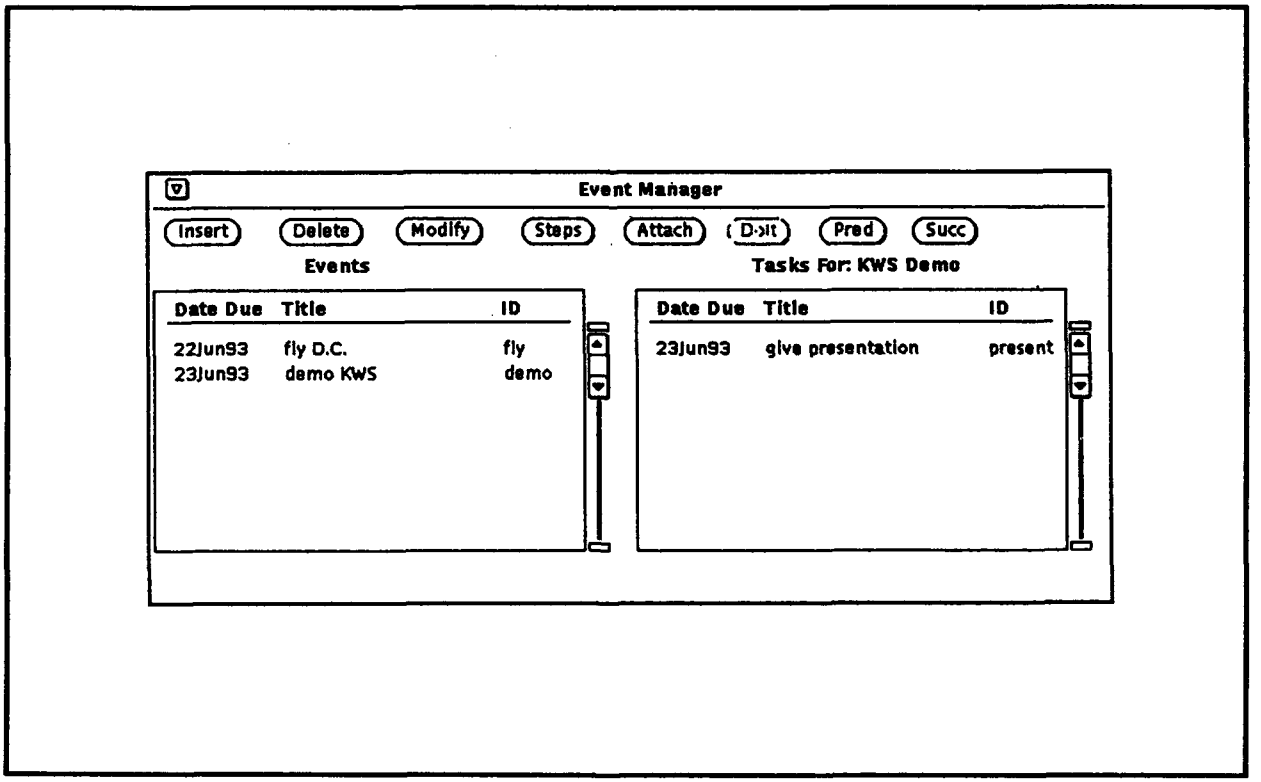


Figure A2. Event manager screen.

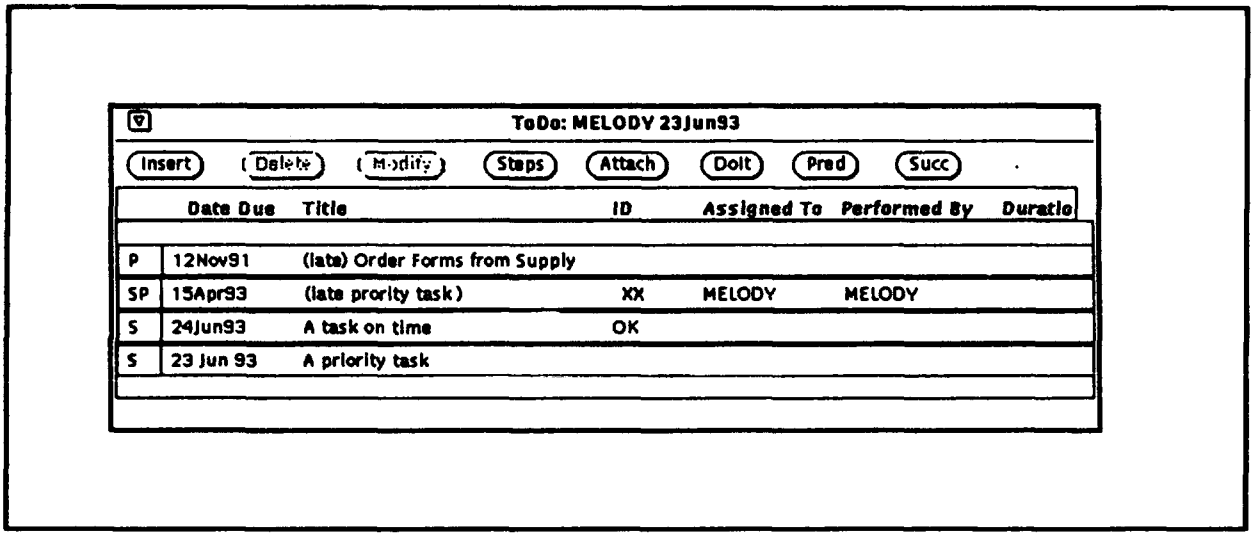


Figure A3. ToDo list screen.

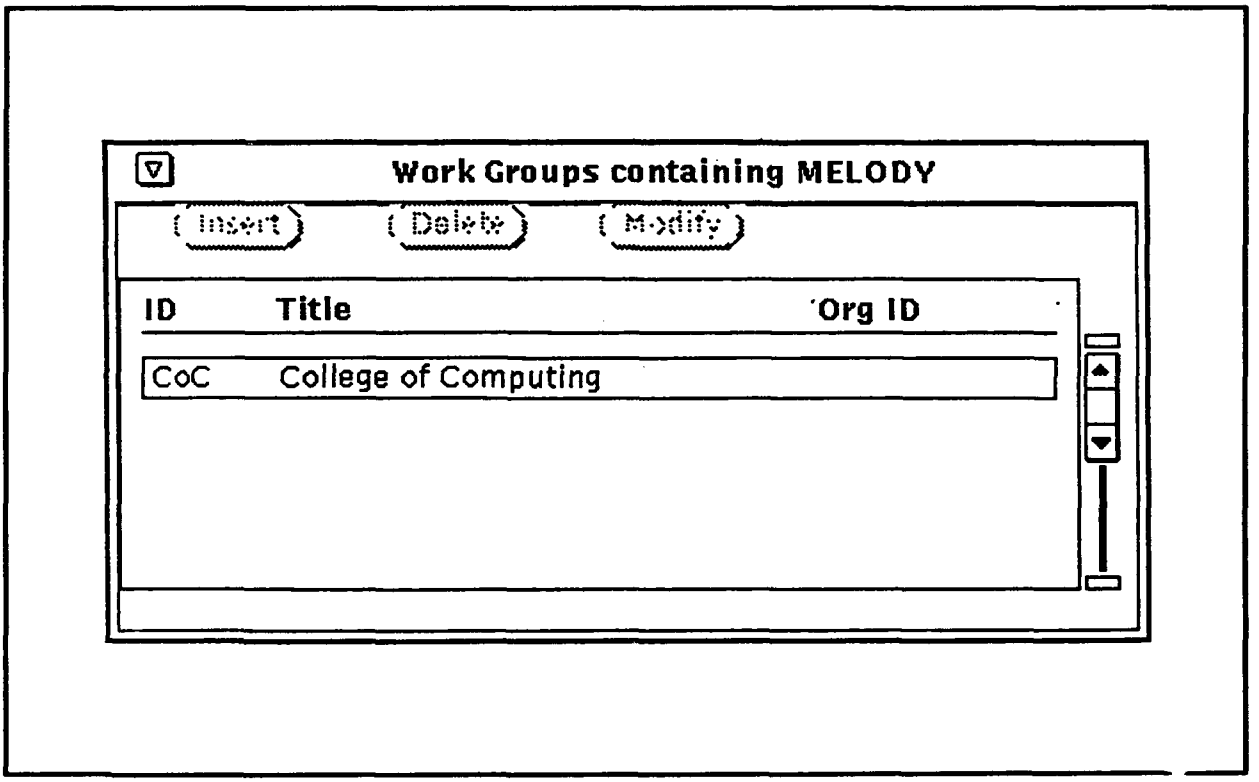


Figure A4. Administrator's screen.

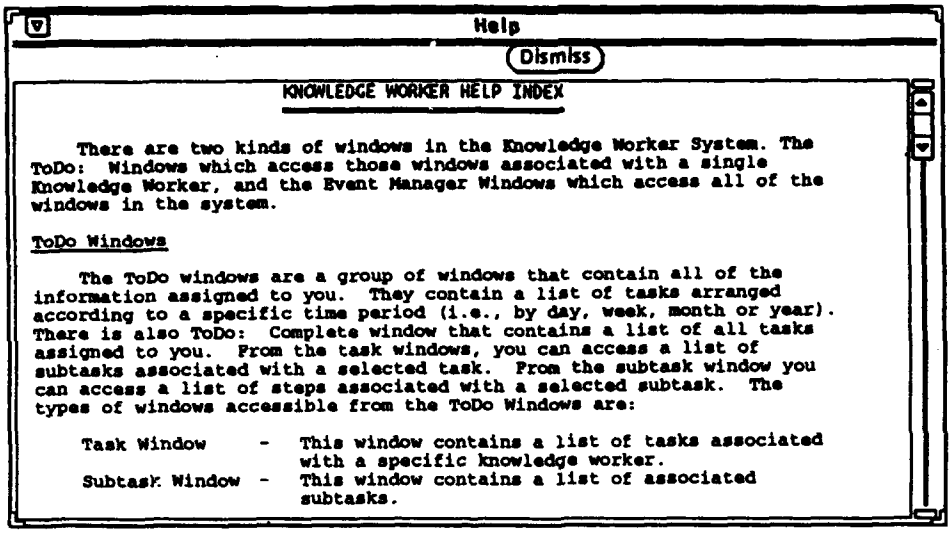


Figure A5. Main help screen.

USACERL DISTRIBUTION

Chief of Engineers
ATTN: CEHEC-IM-LH (2)
ATTN: CEHEC-IM-LP (2)
ATTN: CEIM
ATTN: CECG
ATTN: CERD-M
ATTN: CECC-P
ATTN: CERD-L
ATTN: CECW-P
ATTN: CECW-PR
ATTN: CEMP-E
ATTN: CEMP-C
ATTN: CECW-O
ATTN: CECW
ATTN: CERM
ATTN: CEMP
ATTN: CERD-C
ATTN: CEMP-M
ATTN: CEMP-R
ATTN: CERD-ZA
ATTN: DAEN-ZC

CECPW
ATTN: CECPW-F 22060
ATTN: CECPW-TT 22060
ATTN: CECPW-ZC 22060
ATTN: DET III 79906

US Army Engr District
ATTN: Library (40)

US Army Engr Division
ATTN: Library (13)

HQ XVIII Airborne Corps 28307
ATTN: AFZA-DPW-EE

US Army Material Command (AMC)
Alexandria, VA 22333-0001
ATTN: AMCEN-F
Harry Diamond Lab
ATTN: Library 20783
White Sands Missile Range 88002
ATTN: Library

FORSCOM
Forts Gillem & McPherson 30330
ATTN: FCEN
Installations: (23)

TRADOC
Fort Monroe 23651
ATTN: ATBO-G
Installations: (20)

Fort Belvoir 22060
ATTN: CECC-R 22060
ATTN: Engr Strategic Studies Ctr

USA Natick RD&E Center 01760
ATTN: STRNC-DT
ATTN: DRDNA-F

US Army Materials Tech Lab
ATTN: SLCMT-DPW 02172

SHAPE 09705
ATTN: Infrastructure Branch LANDA

HQ USEUCOM 09128
ATTN: ECJ4-LIE

CEWES 39180
ATTN: Library

CECRL 03755
ATTN: Library

USA AMCOM
ATTN: Facilities Engr 21719
ATTN: AMSMC-EH 61299
ATTN: Facilities Engr (3) 85613

Fort Leonard Wood 65473
ATTN: ATSE-DAC-LB (3)

Military Dist of WASH
Fort McNair
ATTN: ANEN 20319

USA Engr Activity, Capital Area
ATTN: Library 22211

US Army ARDEC 07806
ATTN: SMCAR-ISE

Engr Societies Library
ATTN: Acquisitions 10017

Defense Nuclear Agency
ATTN: NADS 20305

Defense Logistics Agency
ATTN: DLA-WI 22304

Walter Reed Army Medical Ctr 20307

National Guard Bureau 20310
ATTN: NGB-ARI

US Military Academy 10996
ATTN: MAEN-A
ATTN: Facilities Engineer
ATTN: Geography & Envr Engrg

Naval Facilities Engr Command 93043
ATTN: Naval Civil Engr Service Center (3)

USA Japan (USARJ)
ATTN: APAJ-EN-ES 96343
ATTN: HONSHU 96343
ATTN: DPW-Okinawa 96376

416th Engineer Command 60623
ATTN: Gibson USAR Ctr

US Army HSC
Fort Sam Houston 78234
ATTN: HSLO-F
Fitzsimons Army Medical Ctr
ATTN: HSHG-DPW 80045

Tyndall AFB 32403
ATTN: Engrg & Srvc Lab

American Public Works Assoc. 64104-1806

US Army Envr Hygiene Agency
ATTN: HSHB-ME 21010

US Gov't Printing Office 20401
ATTN: Rec Sec/Deposit Sec (2)

Nat'l Institute of Standards & Tech
ATTN: Library 20899

Defense Tech Info Center 22304
ATTN: DTIC-FAB (2)

172
6/94

This publication was reproduced on recycled paper.

☆ U.S. GOVERNMENT PRINTING OFFICE: 1994—3510-S/00037