```
16:34:58        OCA PAD AMENDMENT - PROJECT HEADER INFORMATION              02/15/93

                                                                             Active
Project #: D-48-A43             Cost share #:                           Rev #: 1
Center # : 10/24-6-R7722-0A0    Center shr #:                           OCA file #: 220
                                                                        Work type : RES
Contract#: DACA88-93-D-0003-0001           Mod #: ADMIN.                Document  : DO
Prime    #:                                                             Contract entity: GTRC

Subprojects ? : Y                                                       CFDA: NA
Main project #:                                                         PE #: S22079


Project unit:                DEAN ARCH      Unit code: 02.010.170
Project director(s):
   OLIVE G A                 DEAN ARCH      (404)894-8877



Sponsor/division names: ARMY                               / CON ENG RES LAB, IL
Sponsor/division codes: 102                                / 020


Award period:     921217    to    930630  (performance)    930630  (reports)

Sponsor amount           New this change               Total to date
     Contract value                 0.00                   44,738.00
     Funded                         0.00                   44,738.00
Cost sharing amount                                             0.00

Does subcontracting plan apply ?: Y

Title: FEASIBILITY STUDY OF CONVERTING KNOWLEDGE WORKER SYSTEM


                             PROJECT ADMINISTRATION DATA

OCA contact: William F. Brown          894-4820

 Sponsor technical contact                Sponsor issuing office

 MR. ED JAPEL                             MS. RITA HYER
 (217)352-6511                            (217)373-7280

 US ARMY                                  US ARMY
 P.O. BOX 9005                            P.O. BOX 9005
 CHAMPAIGN, IL 61826-9005                 CHAMPAIGN, IL 61826-9005


Security class (U,C,S,TS) : U            ONR resident rep. is ACO (Y/N): N
Defense priority rating   :  NONE        NA supplemental sheet
Equipment title vests with:   Sponsor X        GIT

Administrative comments -
  ADMIN. REVISION TO DECREASE MAIN PROJECT BY $33,176 TO ESTABLISH SUBPROJECT
  C-49-607/M. EIDBO/CIMR.
```

SE 7532

GEORGIA INSTITUTE OF TECHNOLOGY
OFFICE OF CONTRACT ADMINISTRATION

NOTICE OF PROJECT CLOSEOUT

Closeout Notice Date 08/16/93

Project No. D-48-A43	Center No. 10/24-6-R7722-0A0
Project Director OLIVE G A	School/Lab DEAN ARCH
Sponsor ARMY/CON ENG RES LAB, IL	
Contract/Grant No. DACA88-93-D-0003-0001	Contract Entity GTRC
Prime Contract No.	

Title FEASIBILITY STUDY OF CONVERTING KNOWLEDGE WORKER SYSTEM

Effective Completion Date 930630 (Performance) 930630 (Reports)

Closeout Actions Required:	Y/N	Date Submitted
Final Invoice or Copy of Final Invoice	Y	
Final Report of Inventions and/or Subcontracts	Y	
Government Property Inventory & Related Certificate	Y	
Classified Material Certificate	N	
Release and Assignment	Y	
Other	N	

CommentsEFFECTIVE DATE 12-17-92. CONTRACT VALUE $44,738.

Subproject Under Main Project No.

Continues Project No.

Distribution Required:

Project Director	Y
Administrative Network Representative	Y
GTRI Accounting/Grants and Contracts	Y
Procurement/Supply Services	Y
Research Property Managment	Y
Research Security Services	N
Reports Coordinator (OCA)	Y
GTRC	Y
Project File	Y
Other CARL BAXTER-FMD	Y
FRED CAIN-OOD	Y

NOTE: Final Patent Questionnaire sent to PDPI.

GEORGIA INSTITUTE OF TECHNOLOGY
OFFICE OF CONTRACT ADMINISTRATION

NOTICE OF PROJECT CLOSEOUT (SUBPROJECTS)

Closeout Notice Date 08/16/93

Project No. D-48-A43 Center No. 10/24-6-R7722-0A0_

Project Director OLIVE G A_______________ School/Lab DEAN ARCH______

Sponsor ARMY/CON ENG RES LAB, IL___________________________________

```
Project # C-49-607           PD EIDBO M M                 Unit 02.010.313   T
DO #        DACA88-93-D-0003-000          MOD#                          CIMR  *
Ctr # 10/24-6-R7722-0A1  Main proj # D-48-A43                    OCA CO   WFB
Sponsor-ARMY                             /CON ENG RES LAB, IL             102/020
FEASIBILITY STUDY OF
Start 921217  End 930630  Funded        33,176.00  Contract         33,176.00
```

LEGEND
1. * indicates the project is a subproject.
2. I indicates the project is active and being updated.
3. A indicates the project is currently active.
4. T indicates the project has been terminated.
5. R indicates a terminated project that is being modified.

D 48 - A43

Knowledge Worker Platform Analysis

Final Report

July 23, 1993

Melody Moore - Principal Investigator
Spencer Rugaber
Hernan Astudillo

College of Computing
Open Systems Laboratory
Georgia Institute of Technology

prepared for:
The U.S. Army Construction Engineering Research Laboratory (USACERL)

U.S. Army Contract Number: DACA88-90-D-0040-0010
GIT Project Number: C-49-607

Abstract

The Knowledge Worker System (KWS) is a software application that is designed to help simplify the job of knowledge workers by tracking tasks, associated task information, and schdules. Currently this application is implemented for the PC platform under Microsoft Windows. Since KWS is groupware, it is designed to be used across many computers on a network. It is common today for networks be heterogeneous, with many different machine platforms communicating on the same network. In order to maximize the utility of KWS and to make it more widely available, it should be implemented for other platforms as well as MS-Windows. In order to achieve maximum portability, KWS should be implemented with Open Systems technology, such as POSIX and Motif. This report describes our work analyzing KWS to determine the feasibility of migrating to Open Systems, and a survey of the current supporting tools available on the market. We then describe alternatives and strategies for reengineering the Knowledge Worker system for Open Systems Technology.

Foreword

This research was performed for the U.S. Army Construction Engineering Research Laboratory (USACERL) under Project DACA88-90-D-0040-0010, "Knowledge Worker Platform Analysis". The USACERL technical monitor was Mr. Ed Japel.

The research was performed by the College of Computing, Open Systems Laboratory, Georgia Institute of Technology. Principal Investigator for this work was Melody M. Moore. Also participating in the project were Dr. Spencer Rugaber, Hernan Astudillo, and Terry Kane.

Knowledge Worker Platform Analysis
Final Report

1.0 Introduction

1.1 Problem Statement

Open Systems technology has become an increasingly important issue in computing environments in recent years. In the days when large mainframe computers dominated the computing field, the marketing strategies were centered around vendor competitiveness - developing proprietary systems that were faster, larger, and with more capabilities. With the advent of the personal computer, and then workstations on distributed networks, this competitive philosophy gave way to such needs as interoperability, portability, and usability. The computer industry began to define standards for itself to ensure that software could be developed and used across a variety of hardware, operating systems, and networking technologies.

Following open systems standards has been shown to reduce the overwhelming cost of software development, to improve system reliability, and to reduce maintenance costs [QUA93]. Software applications can be tested for adherence to standards, and therefore we can develop metrics to determine the portability and interoperability of applications.

The Knowledge Worker System (KWS) is by nature a tool to support groups of knowledge workers. Homogenous networks, which are networks that contain only one type of computer, are becoming increasingly rare today. Heterogeneous networks, containing many different computing platforms, are much more ubiquitous. Migrating the KWS to Open Systems would provide portability, maintainability, and interoperability among many different platforms. This report summarizes the feasibility analysis for reengineering the KWS for Open Systems, and presents strategies to implement the migration.

1.2 Objectives

The main objective of the Knowledge Worker Platform Analysis project is to determine the feasibility of migrating the KWS to Open Systems technology, including POSIX and Motif. We also studied the feasibility of reimplementing with the DoD-standard Ada language. We investigated current market availability of language tools, POSIX-compliant operating systems, and Graphical User Interface (GUI) builder tools. Our last objective was to study the transition issues and devise strategic plans, cost estimates, and schedules for the migration of KWS to Open Systems.

1.3 Approach

Our approach included an initial statistical analysis of the code to determine the areas in which we should concentrate our efforts. As a result of the statistical analysis, we determined that the user interface was the largest effort in the reengineering task, and we concentrated on locating user interface tools and support systems for Motif, for both Ada and C / C++ environments. We also surveyed the market for other open systems tools and operating systems. We then used the information from the code analysis and market survey to perform a transition study, concentrating on hardware platform issues, operating systems issues, language issues, user interface issues, and organizational decomposition (object-oriented vs functional) issues.

The version of Knowledge Worker that was analyzed was the most recent version, 1.6. Any changes made after this version will need further study to assess feasibility of migration.

1.4 Document Organization

The organization of the document is as follows:

- The Background section (2.0) describes the Knowledge Worker system's nature and purpose. It then describes Open Systems organizations, standards and conventions. A synopsis of reverse engineering and reengineering techniques is then presented.
- The Vendor survey section (3.0) contains information from our extensive market analysis, which describes the currently available POSIX, Ada, C, and Motif tools.
- The Transition Study section (4.0) details the statistical code analysis, and covers issues in the areas of hardware platform, operating systems, languages, user interfaces, and decomposition issues.
- The Proof of Concept section (5.0) describes the rapid prototype of the Knowledge Worker system developed to show feasibility.
- The Transition Plan section (6.0) presents several possible strategies for reengineering the Knoweldge Worker system, and provides recommendations.
- The References section contain all the bibliographical references from the rest of the report.

2.0 Background

2.1 The Knowledge Worker System

The Knowledge Worker System (KWS) is a software package specifically designed to help simplify the job of knowledge workers [CRC93]. KWS allows knowledge workers to organize and prioritize their work by storing task scheduling information in a centralized database. KWS tracks the scheduled events and any modifications to the schedule. It also serves as a repository of information about each task. KWS helps to 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.

2.2 Open Systems Organizations, Standards and Conventions

Identifying the major standards organizations and their activities is key to understanding the Open Systems world. New developments are constantly occuring in this rapidly developing market, and therefore it is crucial to continuously monitor the journals and newsletters from the various organizations. This section reports on the important standards organizations, relevant standards, and some definitions of conformance.

2.2.1 Open Systems Organizations

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

- **Uniforum** - Uniforum is a non-profit international association of open systems professionals. They publish the Uniforum monthly, a journal of open systems and Unix articles, and Uninews, a biweekly newsletter. Uniforum also publishes annually the 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. (We have joined Uniforum as part of the TRANSOPEN project.)
- **X/Open** - Established in 1984, X/Open is an international independent consortium of computer system vendors with 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 formality, adopting and adapting existing standards as a basis for the CAE. The CAE is being developed through three programs:

  - 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.

  - 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.

  - 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 that are deemed compliant receive the X/Open "brand" that symbolizes its acceptance.

- **IEEE 1003 Committee -** The 1003 series of committees were charted 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 five years, which means that the POSIX.1 standard will be due for review in 1993. [9]
- **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 submitted (international character sets) to submit the POSIX.1 document as a full international standard.
- **NIST -** The National Institute of Standards and Technology originally developed its own operating systems standards, but has since then 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 [11], 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 has been involved with the development of C language standards (ANSI C), that include standard libraries and operating systems interfaces. ANSI is working with the POSIX.1 committee to address these operating-system-specific functions.

2.2.2 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 [14]. Conformance to the standards also ranges from formal certification to partial compliance. This paragraph discusses differences between the standards, and how conformance is measured.

2.2.2.1 Definitions

In order to assess compliance, the formality of the specification must be precisely determined. Therefore, for the purposes of this report, we provide some definitions:

- A **standard** is a formal specification that has been reviewed and approved by a formal standards body, such as ANSI or NIST.
- A **specification** is not necessarily a standard, but may be in the review process to become a standard.
- A ***de facto* standard** is a specification that is not a formal standard that has been approved by a standards organization, but that is so widely used that it is recognized as a standard.
- A **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.
- An **ISP** is an internationally standardized profile.

2.2.2.2 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 1 shows the relevant IEEE specifications and standards:

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

TABLE 1. IEEE POSIX Standards and Specifications

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, 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.

2.2.2.3 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:

- **Strictly conforming** - The application exclusively uses features from the POSIX.1 standard or applicable language standard
- **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 with options and dependencies.

- **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.

2.2.2.4 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 in a Conformance Document.

2.3 Reengineering Methods and Techniques

This section provides an overview of current research in reverse engineering and reengineering systems.

2.3.1 Definitions

First, in order to be clear, we provide some definitions for terms that will be used in this document:

- **Migration** is 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 another, different environment.
- **Reengineering** refers to the task of redesigning and reimplementing code. Reengineering may include changing functionality as well as implementation.
- **Porting** (or **Transporting**) means moving an application from one environment to another with minimal changes. Porting usually implies that nothing more than syntax differences in the code are changed; the resulting porting system should be close to identical to the original system.
- **Reverse Engineering** refers to the process of examining code from an existing application to determine its design.
- **Forward Engineering** is the process of reimplementing a system from a reengineered design.

2.3.2 Migration Strategies

A variety of approaches are conceivable when considering the transition of an information system to a distributed open-systems environment. This subsection lists some of the approaches we have studied. They are organized roughly from those requiring the least to the most effort to effect.

The strategies that are described in this section are comparable in the sense that each has costs and benefits. For any given situation, the costs and benefits for the various strategies must be compared to select the most cost-effective approach. The next subsection describes questions that can be asked in order to make these judgements. This subsection describes the strategies and our experiences with them.

2.3.2.1 As-Is Strategy

The base line against which the other strategies must be measured is the strategy of doing nothing. In this case, there is no real benefit, and the cost is fairly well understood. This strategy may be appropriate if it is known that the application is going to be replaced or phased out. It may also be applicable if the application is used only infrequently by a single site. In this case, there is little value in supporting open systems or distributed access.

2.3.2.2 Direct Application Porting Strategy

Sometimes a system can be reengineered simply by directly porting to the new platform, without adding any new functionality. In order to pursue this strategy, we convert the syntactic problems in the original source code by hand in order to compile it. This is time consuming but relatively straightforward. Once a list of specific conversions can be made, the syntactic conversion can be automated with simple editor macros.

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

2.3.2.3 Conversion by Re-engineering Strategy

Program evolution without benefit of a high-level representation of functionality and structure presents risks in terms of quality. The process of reverse engineering existing software yields such a representation that can then be used as a basis for enhancements.

The benefits of such an approach are obvious; the costs are, however, difficult to measure. One factor that needs to be understood is that reverse engineering requires a significant commitment in time and effort. Some discussions of mechanisms for partially automating the process are described in the next strategy subsection.

The manual re-engineering strategy is indicated in situations where the existing code will continue to be used extensively for the foreseeable future. Maintenance activities that require modification of existing code (versus simply adding new modules) can also help justify the expense of reverse engineering. Reverse engineering does not have to be applied to an entire system. Even if only a part of a system is being reverse engineered, there is still a need for the reverse engineer to understand the context of the component relative to the entire system. Thus, in situations where resources such as accurate documentation or experienced maintenance personal exist, partial reverse engineering may by more feasible.

2.3.2.4 Automatic Reverse Engineering Strategy

Because of the expense involved in reverse engineering, it is desirable to automate as much as possible the steps involved. Unfortunately, the state of the art is such that few tools exist and those that do are capable of describing only surface features of an existing system.

The strategy of automatic reverse engineering involves extracting features from existing programs and translating them into a standard design representation. There are three components of the effort: program analysis, transformation of design information, and design representation and display.

2.3.2.5 Program Analysis

Application programs are highly structured descriptions of computations and data. The programs represent the culmination of a series of design decisions that transform an initial specification into a final program. Moreover, even after a program is delivered, it undergoes subsequent changes for the purposes of removing defects, adding enhancements, and adapting to changing environmental constraints.

In order to construct a high-level design description of a software system, the design decisions that went into its construction and maintenance must be reconstructed. This is accomplished by a systematic analysis of the program text, simultaneously constructing a description of the application domain and procedures that the program models. The analysis can be performed manually, but the process is labor intensive and therefore costly in time and resources. It is desirable to replace as much as possible of the manual effort by the use of automated tools.

2.3.2.6 Transformation of Design Information

Some CASE tools, such as IDE's Software Through Pictures (STP), support reverse engineering users in a variety of ways. In particular, diagrams are stored using a textual representation, and the format of this representation is documented. The normal mode of diagram construction in STP is by the end user manually selecting icons and placing them in the diagram on the screen. Using the published file format, however, we can automatically construct diagrams based on the information extracted by other tools. Then we can forward engineer to a new platform based on this representation.

2.3.2.7 From-Scratch Rewrite Strategy

A final strategy needs to be mentioned for reasons of completeness. Under some circumstances, it may be desirable to throw out the existing program entirely and to rebuild from scratch, including new requirements gathering. This situation may obtain when the old system needs to be significantly modified and its complexity is severe enough that the cost of re-engineering is outweighed by the costs (and risks) of initial development.

2.3.2.8 Decision Criteria

The previous section describes a wide variety of strategies, no one of which is suitable for dealing with all situations. In order to determine which strategy is appropriate in a given situation, those factors that can effect the costs and benefits of applying the strategy should be weighed. These factors are called decision criteria, and this subsection lists and defines them. The next subsection proposes a mechanism whereby the criteria can be organized into a structure that will facilitate the decision making process.

Factors Related to Usage of the Existing System

Usage profile and availability:

- How many users does the system currently have?
- How are these distributed topologically (are they logged into the mainframe, do they submit batch jobs, or are run requests handled manually)?
- How frequently does a given user make use of the application?
- In what different ways are 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 a use of the application is currently made (data entry, validation handled separately; manual or electronic distribution of reports)?
- How many different sites use the existing system?
- Expected lifetime: What is the expected lifetime of the existing application? Is usage growing or shrinking?
- Current execution costs: How much does it currently cost to execute the program in terms of machine and human resources? How does this cost vary across the various types of uses?
- Ownership and control: Are there political factors that would impede the reduction in information control that comes from distributed access?
- Administration: Are there administrative procedures that would be difficult to provide in a distributed environment? What are the costs in transforming these procedures?
- Interoperation: Do other applications depend directly on the data produced by this application (master file, report files, exception files)? Does this application depend on the products of other applications?

Factors Related to the Structure and Functionality of the Existing System

- Current architecture: How amenable is the current architecture to the client/server model? Is the application primarily batch or interactive?
- Hardware configuration considerations - type and resource availability: What external resources and connections does the application require? How extensively are these used?

- Software configuration considerations: Does the existing system make use of non-portable operating system capabilities? Does the existing system interface to other existing systems?
- Reports: Does the existing system write reports? If so, how separable is the computational functionality from the report construction functionality? Are there reports that could be replaced by SQL queries? Are there reports that could be replaced by reports constructed by the RDBMS report writer capability?
- Other RDBMS features: 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 Transited System

- Increased usage: What is the expected increase in usage of the system due to networked availability? What is the expected change in usage (e. g. from batch to interactive) promoted by distributed access?
- DBMS functions: Can the application take advantage of DBMS capabilities such as security and integrity?
- Proposed execution costs: What is the expected change in execution cost in terms of machine and human resources?

Factors Related to Expected Evolution of the Transited System

- Technical impediments: 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?
- Maintenance requirements: How much corrective maintenance activity is there currently on the system? What enhancements to the system are planned? What enhancements would be facilitated by the use of an SQL interface to the data?
- Support issues: Are there personnel available that have experience with the internals of the existing system? Is there existing documentation for the system? How up-to-data and accurate is it? Is sufficient funding available for a comprehensive reverse engineering effort? Does this include funding to support the training of users in 4GLs? How feasible is incremental conversion?
- Standards: Is the application part of the effort to standardize the use of data item names? How closely does it conform to these standards?

3.0 Vendor Survey

An important factor in the feasibility of reengineering Knowledge Worker to Open Systems technology is the availability of tools and resources. This section describes the survey and evaluation of Open Systems and supporting products on the current market.

3.1 Operating Systems

Our primary concern with Operating System software is the level of POSIX compliance. We examined Unix systems for the Sun SPARC architecture and also for the 386 PC architecture.

- 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 x86
  Like SunOS 4.1, the latest release of the Solaris operating system is also POSIX compliant. Solaris 2.0 is not binary compatible with SunOS 4.1.x, however, so care must be taken in choosing application tools to check for implementation on Solaris 2.0. A recent announcement declared that Solaris for PC's will be available mid-July 1993.

- MS-Windows NT (Microsoft)   platform: 386 / 486 and SPARC
  Microsoft's newest 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, Windows NT has been announced for the SPARC platform.

- Santa Cruz Operation (SCO) Unix   platform: 386 / 486
  SCO Unix is certified POSIX-compliant Unix for the PC platform. It is a 32-bit, multithreaded, multitasking, multiuser kernel with virtual memory.

3.2 Ada Compilers

Since C compilers generally are provided with Unix implementations, and the portable GNU C and C++ compilers are public domain, we have concentrated on Ada compilers for this vendor survey.

- 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, Interactive Debugger, Library management system, and other tools. VADS is hosted 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 and will support IEEE 1003.1 chapters 2,4,5, and 6. The release after that is scheduled in December, and will add some low-level features, including Ada IO and signals.

- SPARCWorks Ada (SunPro) List $10,000 Educational price $1,500
  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 (GUI builder for Open Look, which eventually will be rewritten to handle Motif). SPARCWorks Ada can be purchased with the maintenance option that will include the POSIX upgrades this summer and next winter.

- Alsys Ada (Alsys) List $7,500 Educational price $600
  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 ($2250) and to the Xlib and Motif libraries ($2995).

- 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.

3.3 Graphical User Interface Tools

Building a Graphical User Interface (GUI) can be made much easier with GUI builder tools. Some of the toolsets listed below are libraries or widget 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. With the announcement from Sun that Open Look is being discontinued in favor of Motif, we have investigated only Motif-based tools.

3.3.1 Motif Toolkits - C and C++

Motif toolkits - These toolkits are specific to Motif. They 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 Education price $4375
  Reputed to be 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 compile, link, and debug. We have received and installed a demo copy of UIM/X and have found it to be very powerful.

- Builder Xcessory (Integrated Computer Solutions, Inc.)

BX is a tool for building Motif user interfaces with a programming language (C) interface. It also includes a "drag and drop" capability for style sheets. One of our research sponsors, the Army Research Lab, has used BX for Motif development and strongly recommended against using it. Evidently the user interface is cumbersome and the resulting user interface is non-standard.

- Centerline Software (ViewCenter)   list $2995 + $995 for libraries
  This SPARC-based GUI development tool supports OpenLook and Motif. It is basically a GUI builder with hooks to C++. It implements its own toolkit (does not use a "native" toolkit) to have its own "look and feel" for applications.

- C++ Views (Liant Software)   list $1,495 for 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 $2995
  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.

3.3.2 Motif Toolkits - Ada

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

- UIL/Ada and Ada/Motif (SERC)   one copy $2995, 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 $5000 2-5 copies $4500 each
  GRAMMI is an Ada user interface toolkit which supports the development of GUI with X Windows. 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.   (public domain - free)
  The STARS (PAL, formerly SIMTEL-20) repository is a collection of public domain software that can be downloaded from the internet. There is a set of bindings developed by Boeing that can be downloaded that consists of a library of Motif widgets callable from Ada programs.

This is NOT a GUI builder tool - but simply a library. We are investigating the pathnames to obtain these files and will download them. Presumably these bindings will work with a variety of compilers.

3.3.3 Portable GUI Development Toolkits

There are several tools on the market today that are advertised as "GUI Development Tools for portable applications". This means that the designer can write code to a single 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 potentially it seems that we could have one source for KWS that would compile for both Windows and Motif. However, in investigating these tools we have discovered that they have severe shortcomings. Following are the results:

- XVT Portability Toolkit (XVT Software) List $1,450 - $4,400
  The XVT toolkit is advertised to support GUI development for MS-Windows, Macintosh, Motif, Open Look, and character interfaces, among others. It includes a native toolkit and a GUI builder (a WYSIWYG "palette" tool). We spoke with developers who had used this tool to develop an application that had both MS-Windows and Motif user interfaces. They strongly recommended *against* using this tool. 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 the tool advertises that the programmers only need to use a single API, that it was necessary to go into the generated code to customize and fix problems, causing more development time than necessary. This group now has a sizable investment in the XVT tool, but they are considering starting over from scratch and developing two separate interfaces, using Windows-specific and Motif-specific toolsets (and having two copies of the source). They felt it would save a lot of development time and frustration, and would allow them to be independent of a single vendor (all of their code is locked in to using the XVT tool's Application Programming Interface, it cannot be ported to another GUI builder tool).

- Open Interface (Neuron Data) List $7,000 - $15,000 developers
  Supports: Motif, Open Look, Windows, PM, 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 it enables the company to produce a more flexible product than if it stuck with the native toolkits. We suspect that this tool also diverges from the standards, especially because of the proprietary implementations of the toolkits.

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

3.4 Database Access

The current implementation of KWS is done with a centralized Oracle server. Since SQL is a standard 4GL, it could be generalized for other database server programs. However, we assume that Oracle will be retained for the server. The following tools are provided for application programs to interface with Oracle servers.

- Pro-Ada (site licensed by Georgia Tech)
  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 (site licensed by Georgia Tech)
  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.

4.0 Transition Study Results

This section describes the transition study, which examined issues concerning platform, operating system, user interfaces, language, and organizational decomposition.

4.1 Statistical Analysis

In order to gain insight into the nature of the KWS application, the first step was to perform a statistical analysis of the source code. This allowed us to assess the relative importance of these issues according to amount of code devoted to each of the areas of study, and to determine which areas would most affect the reengineering effort. We obtained the KWS source code for the MS-Windows version 6.0, and used a combination of several techniques to glean statistical information about the application:

- **Inspection and Analysis** - The most tedious and labor-intensive way to learn code function, this is accomplished by simply reading code and comments. This method is used to make subjective judgements, such as code and comment quality.
- **Developer interviewing** - The reengineering process is made considerably easier if the original developers of the candidate system are available for interviewing. We questioned the KWS system developers for their estimates of complexity and areas of difficulty.
- **Automated tools** - Automated tools can quickly and efficiently give answers to statistical questions that could take hours if done by hand. We made extensive use of the Unix tools *grep* (global regular expression parser), *wc* (word counter) and *diff* (file comparison) to examine the source code for occurrences of system calls, interfaces to databases, and other statistics.

4.1.1 Statistical Analysis Results

Figure 1 below shows the initial analysis of the code from the automated tool method. The number of Lines of Code (LOC) allow us to classify Knowledge Worker as a medium-sized application. Knowledge Worker depends on two interfaces: the MS-Windows Application Programming Interface, which implements the graphical user interface, and Oracle, the database interface. We examined these areas to determine how much of the code is platform-specific and therefore will need to be rewritten. Figure 1 shows the initial analysis of the code, done mostly with automated tools.

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

Figure 1 - Intital Analysis of Code

The next step was to examine the code to determine percentages that might give us information on the level of difficulty for migration. Figure 2 shows the distribution analysis of the code:

User Interface Code:		85%
Algorithmic Code:		
	Scheduling module	2%
System Interface Code:		
	Database Access	9%
	File I/O	3%
	Process interface	1%

Figure 2 - Code Distribution

4.1.2 Conclusions from Statistical Analysis

This revealing analysis shows us that the clear majority of the code is in the user interface. Therefore, the largest part of the reengineering effort will center on rewriting the MS-Windows based graphical user interface to conform to X Windows and Motif functionality. Due to the differences in MS-Windows and Motif, this will probably entail some redesign as well as reengineering.

The next most significant piece of the Knowledge Worker code is the database (Oracle) access code. This code may be easier to reengineer than the user interface because it is likely that the actual SQL calls will remain the same. Therefore, the reengineering task would 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 reengineered because of the substantial differences between the MS-Windows operating system and the Unix/POSIX operating system. The remaining algorithmic code (2%) is the only code that probably could be used as-is in a reengineering to Open Systems.

In summary, the large majority of KWS code is platform-dependent and therefore will have to be reengineered for the Open Systems environment.

4.2 Hardware Platform Issues

The largest issue in platform dependence is the availability of tool support and the differences in operating systems. We have thoroughly examined the tool issue (see report above and monthly report from April 1993 for full summary of tool availability). We have also considered the issues inherent in Operating System differences, and the implications of the POSIX standard on our development. In studying the Knowledge Worker code, we have not found any hardware dependencies outside of those handled by the Operating System. Therefore, we do not anticipate platform-dependent problems that are not already addressed by the Operating System conversion.

Since the Sun SPARC platform is the largest-distribution workstation, and the most comprehensive set of development tools exists for this platform, we will perform the initial reengineering to POSIX and Motif on the SPARC. In a later phase of the project, we will perform a true port to a totally different ubiquitous architecture, the 386 / 486 PC.

4.3 Operating System Issues

This section details issues that arise in reengineering from MS-Windows to the Unix/POSIX environment. Part of this study entailed attempting to devise mappings from Windows capabilities to POSIX features. According to the statistical analysis of the code, the operating system-dependent portion comprises approximately five percent of the system. Other than the services mapping described below, the only issues are differences in the filesystems. Unix file names are case-sensitive, while MS-Windows filenames are not. There are also syntactic differences in the filenames that must be taken into account.

4.3.1 Operating System Services Mapping

In order to determine the feasibility of supporting all of the KWS functionality in open systems technology, we examined the amount of MS-Windows operating system calls [REC92], and attempted to map these calls to the corresponding POSIX system calls defined in IEEE 1003.1

[IEE88]. We were able to map all of the OS-specific calls to POSIX calls, so all of this functionality can be supported with open systems. Following is the mapping of KWS MS-Windows operating systems services and the POSIX calls that fulfill the functionality:

- 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 the KWS application on MS-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 in our proof of concept we experimented with mappings from MS-Windows child window types to Motif widgets.

- The implementation of KWS does NOT utilize 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).

4.4 Language Issues

The current implementation of KWS for MS-Windows is written in C. However, it has been shown that only 2% of the code (the algorithmic scheduling module) could potentially be ported directly. Since the great majority of the code must be reengineered, the language issues then center mostly on 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. The C language is widely available, and there exists an ANSI standard for the language. If ANSI C is adhered to, then C, is fairly portable (there is a standard Unix tool, *lint,* that can evaluate conformance of source code to ANSI C).

• Compiler availability

Ada compilers are now available for almost every hardware platform, but they do tend to more expensive than C compilers. Some POSIX-compliant compilers are available, and more are scheduled to be on the market soon. C compilers and libraries are usually provided with Unix distributions, and good quality public domain C compilers are available free of charge (the GNU toolset). 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)

Our market survey showed that there are GUI builders available for both C and Ada, with slightly more tools available for C. Some of the tools produced C, which then could be turned into Ada through a translation step. There are public domain Motif bindings 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 such as pragmas are avoided, then code written in Ada has been shown to be very portable. C is also portable, and compilers for the language are ubiquitous. However, C has many more possibilities for divergence than Ada. If the C language 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 obviously exist. Recently, IEEE 1003.5, Ada language bindings to POSIX, were approved. Market vendors have responded and several POSIX-compliant Ada compilers will be available by summer 93.

4.5 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 Operations) [UNI93] detailed an effort for these vendors to cooperate on developing a Common Open Software Environment (COSE, pronounced "cozy"). 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 the user inteface to the desktop will be standardized. This announcement 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 (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 we will by default now be reengineering for Motif. Since we know from the statistical analysis that 85% of the KWS code is devoted to the user interface, this is a primary area of concern. For this reason, we chose to prototype the user interface for our Proof of Concept (described in section 5.0). Because of the differences between MS-Windows and Motif, the user interface will need to be reengineered. Some small changes in the appearance of the user interface will be necessary. These are detailed in the Proof of Concept section.

5.0 Proof of Concept

This section describes the rapid prototype of the user interface that was built to show the feasibility of using open systems Graphical User Interface tools to reengineer the Knowledge Worker System. Since the Knowledge Worker source code is 85% user interface code, the user interface is the most important component to reengineer to assess the difficulty of the migration. Appendix A contains graphical representations of the screens generated for the prototype.

5.1 Overview

The prototype of the KWS user interface is intended to illustrate representative paradigms, differences, and problems in reengineering from MS-Windows to Unix and Open Systems. For reasons of expediency and availability, this prototype was built using the Sun tools Dev/Guide and the Open Look toolkit. The course followed was to translate the interface in an item-by-item fashion, and document our assumptions and the resulting transformations.

5.2 Transformations and Assumptions

The fundamental problem in the translation of an interface into another (while preserving its functionality) is the different stylistic conventions. For example, Open Look (OL) does not have menu bars, which are present in other toolkits; and OL applications don't have "quit" options, because this is handled by the window manager. The best that can be done is to render the functionality and "look and feel" as close as possible, while respecting the conventions in the receiving end.

The main concerns have been (in order of resolution):

- the item functionality, and
- its appearance.

For example, a Windows menu that shows a menu when pressed must be mapped to a OL button that shows a menu when pressed and has similar in 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 doesn't have a direct correspondence, because of different button sizes or of alignments.

5.2.1 Mapping the Interface

The translation of each item has 3 steps:

1. Determine the equivalent Open Look item to correspond to the MS-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)

Figure 3 below characterizes the mapping that was used for the prototype.

WINDOWS ITEM	**DEVGUIDE ITEM**
[a1] menubar	[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	[h2] menu item
... not selectable	... set "Inactive"
[d1] submenu 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)

OTHERS:

The Attachment window has been translated into a TextPane, which loads a file when opened; this corresponds to the exact behavior of the KWS.

Figure 3 - Windows - Devguide Mapping

6.0 Transition Plan

This section outlines the choices and alternatives available for devising a strategy to fully migrate the Knowledge Worker System to Open Systems technology, and to perform a validation step to assess its portability.

6.1 Platform choice

As described in the Transition Issues section for Platform, the Sun SPARC architecture is the recommended choice for the first reengineering effort. This platform was chosen because it supports the best set of development tools currently on the market. Once the reengineering effort is complete, then the Open Systems version of Knowledge Worker can be ported to other POSIX-compliant architectures. We propose to port the resulting system 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.

6.2 Strategic Alternatives

6.2.1 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 reengineered to the proprietary language of the tool, then code could be automatically produced for each of the different user interface technologies. This would allow one source to be maintained that would produce code for MS-Windows, Motif, Open Look, and even MacIntosh. However, upon further study of these tools, and experience reports from large development projects that have used them, we discovered some serious flaws:

- The developer would become locked into a proprietary intermediate language. This is dangerous for several reasons - the vendor has total control over the representation of the language and could change it at their discretion, causing major rework. Also, using a proprietary language violates the spirit of Open Systems.
- The tools are not robust or precise enough to completely specify the interfaces, therefore necessitating changes in the generated code to achieve the desired effects. This would by its nature create divergent sources, making the interface builder tools nearly useless.
- The SQL interface might differ on different hardware platforms, necessitating divergent sources in this area as well.
- In general, the interfaces generated by these tools are inferior. As one would expect, the "jack of all trades, master of none" adage applies here. The interface technologies supported by these tools vary widely enough that no one tool can currently support all of them well.

In light of these drawbacks, the advantage of potentially having one single source for all platforms is negated. Therefore, this strategy is not recommended at this time.

6.2.2 Alternative 2 - Motif-Specific GUI Tool

The other alternative is to maintain two separate sources, the existing one for MS-Windows, and the newly reengineered Open Systems source using a Motif-Specific GUI builder tool. The palette-based tools available currently are quite adequate and can significantly enhance the development process. Since there are now tools that can support Motif, POSIX compliance, and Ada, this is the recommended strategy for reengineering the user interface.

6.2.3 Language issues

Part of determining the feasibility of reengineering KWS to Open Systems technology was to assess the state of tool support for Ada. The findings indicate that adequate tool support for Ada development does indeed exist. Since Ada is the standard DoD language, this is the recommended strategy, bearing in mind the following considerations:

- Tools are currently available, although the tool choices are rather limited and the tools are also relatively expensive.
- Ada technology for open systems is still nascent, and some of the reengineering work may need experimentation to solve as-yet-unknown problems.
- Since the original KWS is written in C, the Ada implemention will totally diverge from the original, there will be no code sharing. All modifications to future versions of KWS will need to be done to both the original and the reengineered Ada versions of the software.
- The expertise pool for developing and maintaining Ada applications is more limited than the expertise pool available for C.

We studied C as an alternative, and although Ada is the recommended choice because of the DoD standard, a reengineering in C would also be feasible and actually would be a simpler effort. There are some advantages that C would offer:

- Some of the code (albeit a very small amount, about 2 - 5%) of the code would not have to undergo a reengineering process, and could be directly used.
- Using C would reduce the amount of experimentation necessary, and therefore would reduce risk in cost and schedule estimation.
- C tools are widely available, often in the public domain.
- The interfaces for open systems (notably POSIX and Motif) are defined in C, therefore they are the most well-tested and the most available.

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

6.3 Strategy Recommendation

This section details the recommended strategy for reengineering the Knowledge Worker System to Open Systems technology and Ada. Included in this section are an overview of the development strategy and an associated manpower estimate, development schedule, and a cost estimate for personal services, equipment, and tools.

6.3.1 Development Strategy

The reengineering effort should be performed in two phases; an initial reengineering phase, choosing a development platform that has the best tool support for redesigning and reimplementing the KWS application. According to our vendor and tool survey, the Sun SPARC platform has the best development environment available for open systems tools. The next phase of effort is a true port, migrating the 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, this is the recommended path for the first migration, since SCO Unix is POSIX-compliant and is available for the 386/486 PC platform. Subsequent migration platforms can be included as needed.

6.3.2 Manpower estimate

Following are the list of tasks associated with the initial reengineering of the KWS and the port and validation phase. An estimate of effort is provided for each task.

Phase I - Reengineering

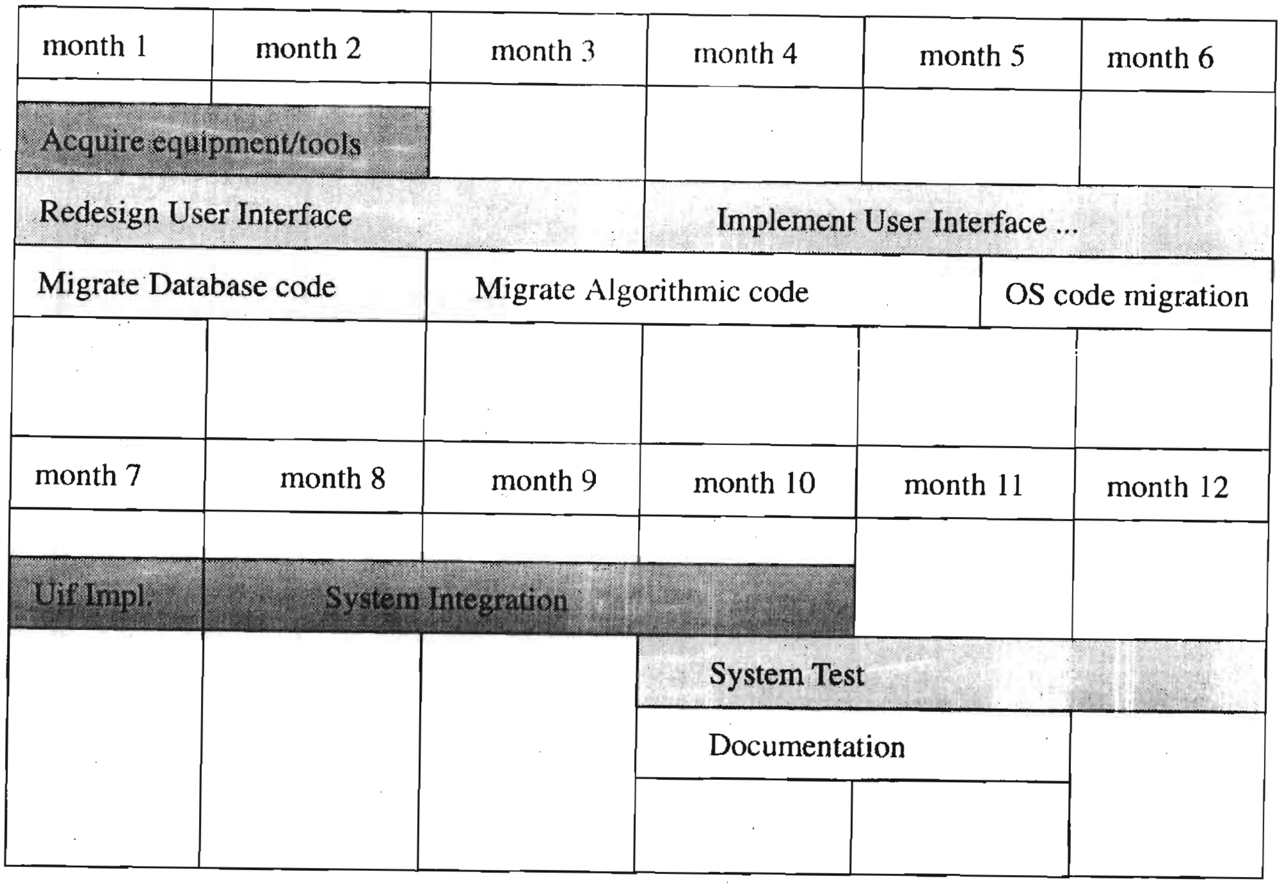
Task	**Personweeks**
Acquire/install/learn equipment and tools	8
Redesign User Interface	12
User Interface Implementation	16
Reengineer database access code	8
Reengineer algorithmic code	10
Reengineer operating system interface code	8
System Integration	12
System Test	12
Documentation	8
Project Administration and management	16
Technical direction	12
Total effort	**122 personweeks**

Phase II - Port and Validation

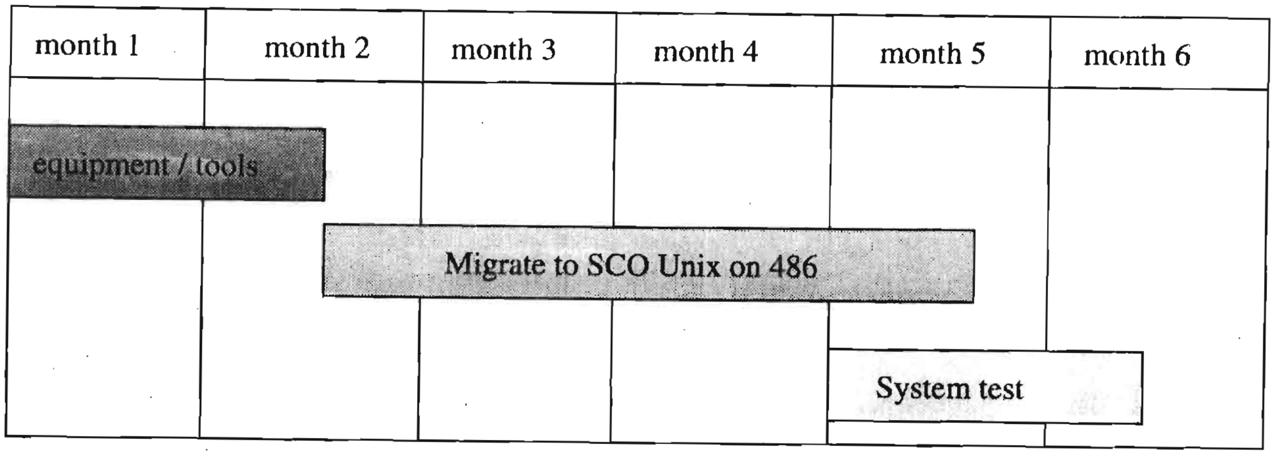
Task	**Personweeks**
Acquire equipment and tools	6
Port to SCO Unix on 386/486	16
Total effort	**22 personweeks**

6.3.3 Schedule

Following is a modified Gantt chart showing the projected completion schedule for Phase I of the KWS migration:



Following is the Gantt chart for the completion of the Phase II migration:



6.3.4 Cost Estimate

Following is a summary cost estimate for both Phase I and Phase II of the KWS migration:

Personal Services			**Totals**
RS-II Project Manager	@$25.38/hr	1392 hours	$35,333.33
Senior RS consultant	@$34.96/hr	348 hours	$12,166.67
RS-I Programmer	@$20.11/hr	4,524 hours	$91,000.00
Fringe @25.1%			34,763.00
Materials and Supplies			24,765.00
UIM/X GUI Toolkit	4,375.00		
UIL/Ada and Ada/MOTIF	3,890.00		
SCO Unix for 486	1,000.00		
Alsys Ada - SPARC	7,500.00		
Alsys Ada - 486	7,500.00		
Supplies	500.00		
Travel (4 quarterly reviews)			3,000.00
Computer Charges			11,229.00
Total Direct			212,257.00
Overhead @37%			78,535.28
Equipment			31,795.20
SPARC 10 model 41 (2 @ 14,556)		29,113.60	
Media, documentation, cables for SPARC		1,081.60	
486 PC		1,600.00	
Total Cost Estimate			**322,587.00**

7.0 References

[CRC93] Construction Research Center, Knowledge Worker System Version 1.60 User Manual, Georgia Insittute of Technology. Prepared for the U.S. Army Construction Engineering Research Laboratory, Champaign, Illinois, April 1993.

[IEE88] IEEE, [9] POSIX 1003.1 Specification (ANSI Standard). Institute of Electrical and Electronics Engineers Inc, 1988.

[IEEE2] IEEE 1003 Committee. Technical Standards Reference Model, international standard 1003.3.

[HUM92] NTIS, Human Computer Interface Style Guide, NTIS Accession number ADA 253475 38.92.

[NIS91] NIST, Application Portability Profile, The U.S. Government's Open System Environment Profile OSE/1, Version 1.0, April 1991.

[QUA93] Quarterman, John, and Wilhelm, Susanne. Unix, POSIX, and Open Systems, Addison Wesley Unix and Open Systems series, 1993.

[REC92] Rector, Brent E. Developing Windows 3.1Applications with Microsoft C/C++, Second Edition, Sams Publishing, 1992.

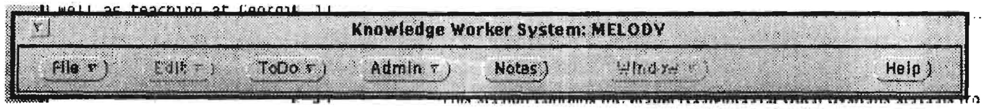
[TRM92] DoD Architecture Implementation Concept for Information Systems. Techical Reference Manual, Version 1.3, January 1993.

[UNI93] 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, March 17, 1993.

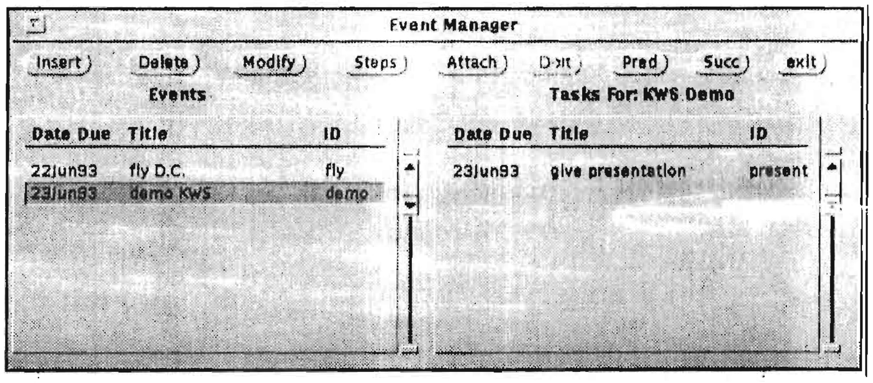
Appendix A - Prototype Screens

This section contains some of the representative reengineered Open Systems screens from the prototype. This shows the difference in look-and-feel in the Open Systems user interface.The prototype was developed using Sun's Dev/Guide palette-based GUI builder tool on a SPARC platform.

Main Screen



Event Manager Screen



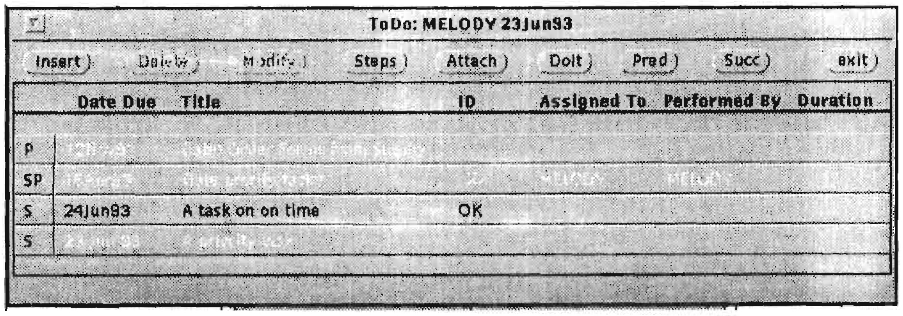
Events

Date Due	Title	ID
22Jun93	fly D.C.	fly
23Jun93	demo KWS	demo

Tasks For: KWS Demo

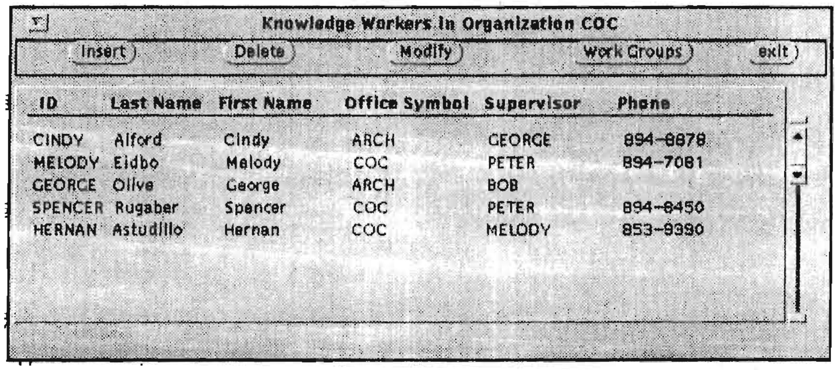
Date Due	Title	ID
23Jun93	give presentation	present

ToDo List Screen



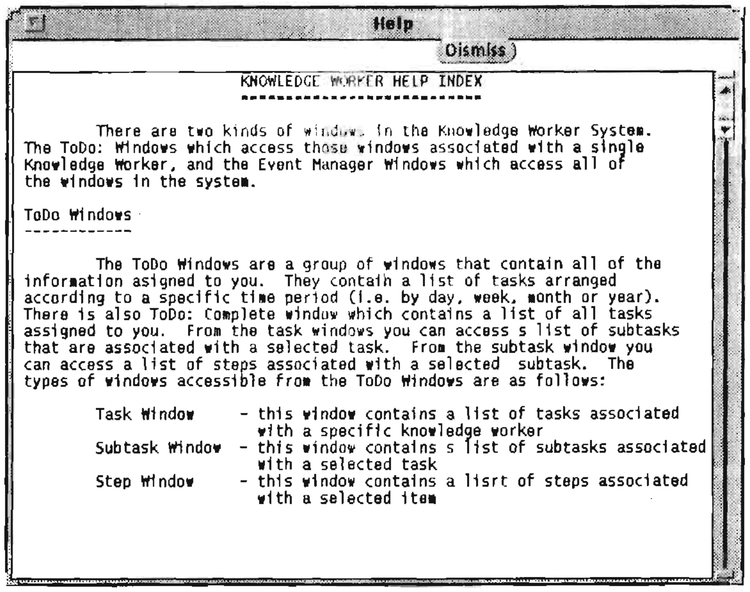
	Date Due	Title	ID	Assigned To	Performed By	Duration
P	[illegible]	[illegible]				
SP	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	
S	24Jun93	A task on on time	OK			
S	[illegible]	[illegible]				

Administrator's Screen



ID	Last Name	First Name	Office Symbol	Supervisor	Phone
CINDY	Alford	Cindy	ARCH	GEORGE	894-8878
MELODY	Eidbo	Melody	COC	PETER	894-7081
GEORGE	Olive	George	ARCH	BOB	
SPENCER	Rugaber	Spencer	COC	PETER	894-8450
HERNAN	Astudillo	Hernan	COC	MELODY	853-9390

Main Help Screen



```
                     KNOWLEDGE WORKER HELP INDEX
                     ===========================

        There are two kinds of windows in the Knowledge Worker System.
The ToDo: Windows which access those windows associated with a single
Knowledge Worker, and the Event Manager Windows which access all of
the windows in the system.

ToDo Windows
------------

        The ToDo Windows are a group of windows that contain all of the
information asigned to you.  They contain a list of tasks arranged
according to a specific time period (i.e. by day, week, month or year).
There is also ToDo: Complete window which contains a list of all tasks
assigned to you.  From the task windows you can access s list of subtasks
that are associated with a selected task.  From the subtask window you
can access a list of steps associated with a selected  subtask.  The
types of windows accessible from the ToDo Windows are as follows:

        Task Window     - this window contains a list of tasks associated
                          with a specific knowledge worker
        Subtask Window  - this window contains s list of subtasks associated
                          with a selected task
        Step Window     - this window contains a lisrt of steps associated
                          with a selected item
```