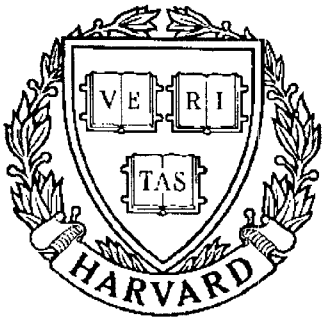
THESIS REPORT
Master's Degree

SYSTEMS
RESEARCH
CENTER



Supported by the National Science Foundation Engineering Research Center Program (NSFD CD 8803012), the University of Maryland, Harvard University, and Industry

Systems Engineering An Interactive Software Tool for Creating Interactive Graphical User Interfaces

by D.D. Opiekun
Advisor: O.A. Asbjornsen

M.S. 91-6

SYSTEMS ENGINEERING AN INTERACTIVE SOFTWARE TOOL FOR CREATING INTERACTIVE GRAPHICAL USER INTERFACES

by

Deborah Dorrelle Opiekun

Thesis submitted to the Faculty of the Graduate School
of The University of Maryland in partial fulfillment
of the requirements for the degree of
Master of Science
1991

Advisory Committee:

Professor Odd A. Asbjornsen, Director/Advisor
Professor Alan Hevner
Professor Ben Shneiderman

ABSTRACT

Title of Thesis:	SYSTEMS ENGINEERING AN INTERACTIVE SOFTWARE TOOL FOR CREATING INTERACTIVE GRAPHICAL USER INTERFACES
Name of Degree Candidate:	Deborah Dorrelle Opiekun
Degree and Year:	Master of Science, Systems Engineering, 1991
Thesis Directed by:	Dr. Odd A. Asbjornsen, Director Systems Engineering Program, Systems Engineering Program

This thesis presents the Systems Engineering Life Cycle and its application to the development of an interactive software tool, XMISE. This software tool allows users to interactively create and specify interactive graphical user interfaces for X Window System- and OSF/Motif-based software applications. This thesis describes each phase in the Systems Engineering Life Cycle and details how each phase was performed during the development of XMISE, including the Computer-Aided Systems Engineering (CASE) tools employed, the required team interaction, the outputs produced, the reviews held, and the lessons learned. Examples of the different types of documentation produced during each life cycle phase are included.

By following the Systems Engineering Life Cycle and applying principles of Systems Engineering, a software tool was produced that meets customer and user requirements, solves the stated needs, and is documented with requirements traced from analysis through implementation and testing.

ACKNOWLEDGMENTS

I would like to give special thanks to Gary L. Moxley, who participated in and contributed equally to the entire Systems Engineering Life Cycle as part of his thesis project for Hood College; Jerry L. Wernimont and Dawn Ramsburg for their never-ending support, creativity, and suggestions throughout the development of XMISE; Chris King for her management support as the industry advisor; Jan Campanaro for her clerical support; and Jeffrey W. Opiekun, my husband, for his patience, support, and understanding.

TABLE OF CONTENTS

LIST OF FIGURES

INTRODUCTION

This thesis presents the Systems Engineering Life Cycle and its application to the development of an interactive software tool. The X and Motif Interactive Screen Editor, XMISE, allows users to interactively create and specify interactive graphical user interfaces for X Window System- and OSF/Motif-based software applications. Each phase in the Systems Engineering Life Cycle was performed, including formulating and determining operational needs, defining a system concept, planning the system management, defining and analyzing system requirements, designing the system concept, developing the system design, and producing, deploying, and evaluating the developed system. This thesis describes in detail how each phase was performed during the development of XMISE, including the Computer-Aided Systems Engineering (CASE) tools employed, the required team interaction, the outputs produced, the reviews held, and the lessons learned. Samples of the different types of documentation that were produced during each phase in the life cycle are included.

XMISE was developed for COMSAT Laboratories under its 1990 Proprietary Research Program and as a joint Master's project for graduate students in the Systems Engineering Program at the University of Maryland and the Computer and Information Sciences Program at Hood College. Systems Engineering is a discipline that emphasizes working as a team to satisfactorily solve a customer's stated need or problem. The XMISE project provided a means to demonstrate that students from two different Master's programs at two different schools could team together with their respective advisors, the customers, internal reviewers, and the department

manager; follow the Systems Engineering Life Cycle; and produce a usable software tool that met the customers' requirements. In addition, it provided the students with an opportunity to participate in all of the activities of the Systems Engineering Life Cycle.

System Engineering Life Cycle and Goals

Systems Engineering is an iterative, top-down, controlled process by which users' needs are understood and transformed into an operational system (Asbjornsen 1988; Eisner 1988). Every system has a life cycle from birth to death. Different literature provides different names for the phases within the life cycle; however, they all include similar activities. Figure 1 illustrates the division of the Systems Engineering Life Cycle into phases that occur over time. These phases include determining and formulating operational needs; defining a system concept; planning and managing the system; defining and analyzing the system requirements; designing the system to meet the requirements; building the system according to the system design; producing, deploying, and evaluating the developed system; operating, supplying, maintaining, and upgrading the system; and finally retiring the system. (Asbjornsen 1988; Blanchard and Fabrycky 1981). It also illustrates that many of the phases are iterative, and that there may be iteration between phases. Because XMISE is a new system, the last two phases will not be addressed in this thesis except in the context of considering these phases in designing the system.

Figure 1 also illustrates the major Systems Engineering products that result from performing each phase in the life cycle. These products are the minimum set of products that may be produced. The Systems Engineering Management Plan

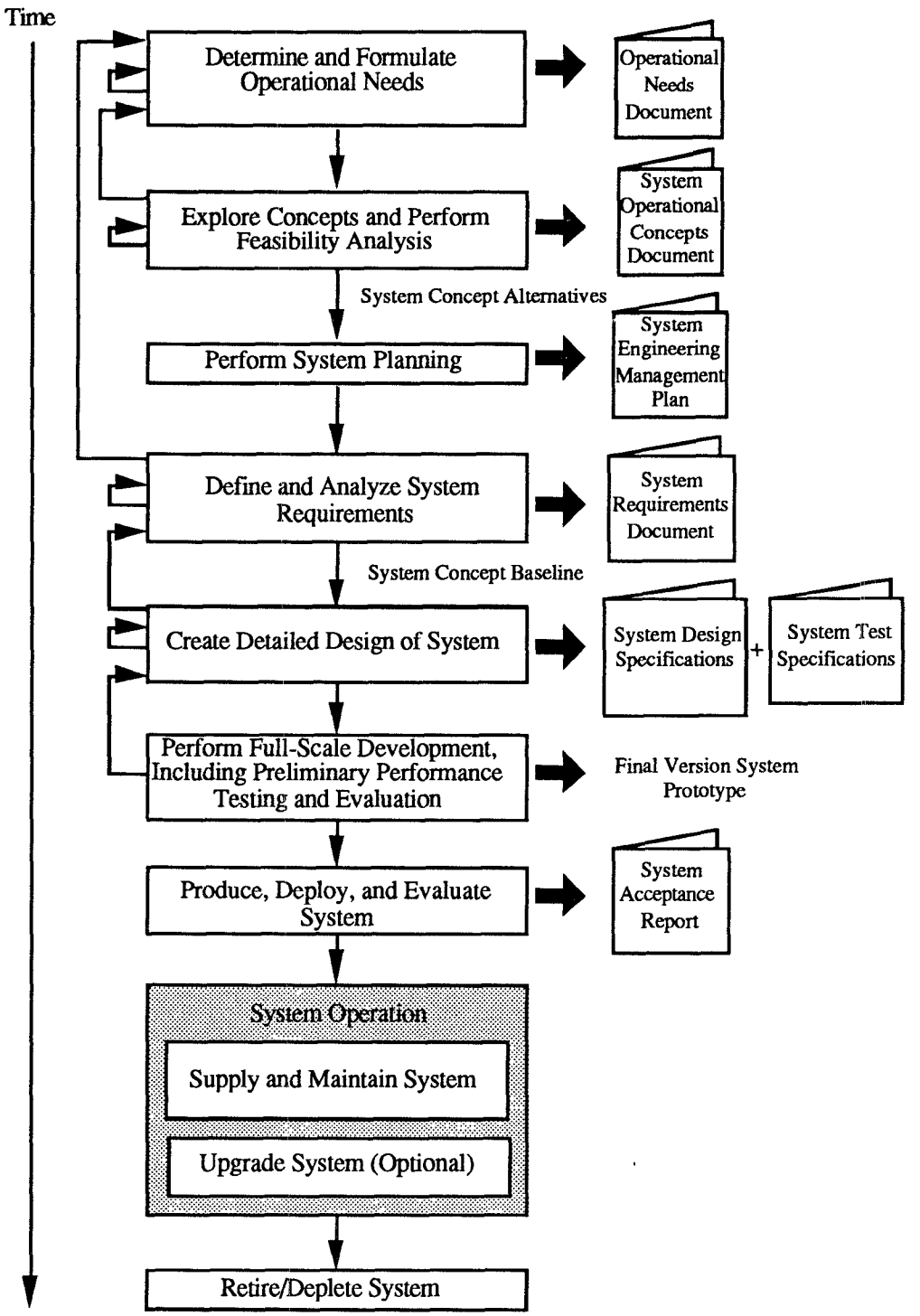


Figure 1. The Systems Engineering Life Cycle

(discussed in Chapter 3) and the system requirements may identify several other products that must be produced during each phase.

The goal of Systems Engineering is to achieve customer and user satisfaction without cost and schedule overruns, gold-plating, failures, deficiencies, omissions, or solving the wrong problem. In addition, these goals are to be achieved with minimum adverse effects to humans and the environment (Atallah 1989).

Background of the XMISE Project

Before determining the operational needs of a project, the customer's environment must be understood and documented. For the XMISE project, interviews held with the customers produced a detailed description of their environment, and revealed that COMSAT developed analytical software applications on IBM and DEC mainframes as well as on IBM personal computers. Typically, these applications were command-driven, but users were increasingly demanding interactive user interfaces to these applications. COMSAT found that interactive graphic applications were difficult to develop on these platforms and performed only at satisfactory levels. Thus in 1989, COMSAT began a proprietary research task to build a User Interface Management System (UIMS) on an ULTRIX-based workstation.

In 1990, the UIMS included a sophisticated, "what you see is what you get" (WYSIWYG) text editor, a MacDraw II library from which Motif widgets could be easily drawn to create user interface screen designs, and a toolkit containing C functions that simplified the interface to OSF/Motif and the X Window System. It also included a working prototype of XMISE which allowed a user to interactively create and save an application's user interface, that was based on the X Window

System and OSF/Motif in User Interface Language (UIL) code. The goal of the XMISE project was to replace the prototype version of XMISE with a system that provided full functionality, was well-documented and maintainable, and would be incorporated into the UIMS.

The description of the customers' environment was documented in the XMISE Operational Needs Document, and repeated in the XMISE Requirements Document and the XMISE Screen Specification. Documenting the current environment identified several project constraints. For example, since XMISE was to become part of the existing UIMS, it had to adhere to the styles and conventions set forth for the UIMS, as well as interface with the existing UIMS components.

CHAPTER 1. OPERATIONAL NEEDS DETERMINATION

Determining the operational needs for a system is the first phase in the Systems Engineering Life Cycle as shown in Figure 1. A customer typically has a need "based on a want or desire for some item(s) arising out of a perceived deficiency" (Blanchard and Fabrycky 1981). This need usually takes the form of a broad explanation of why the system is necessary and what specific needs are to be satisfied (Eisner 1988). From this needs statement, the systems engineer, assisted by the customer, must address the following questions in order to understand the customer's goals, desires, and concept for the system (Asbjornsen 1988; Eisner 1988; Blanchard and Fabrycky 1981):

- What is the system's purpose, function, or goal?
- How will the system provide this function or accomplish its goal?
- How well will the system perform?
- Who will operate the system and how will the system be operated?
- How will the system be maintained?
- What are the system constraints?
  - When is the system needed and for how long?
  - What is the system's environment?
  - What is the proposed budget?
  - Who is available to build the system?

The answers to these questions will identify the operational needs for the system. As each need is identified, its impact on previously identified needs must be

determined. As a result, determining the operational needs for the system is an iterative process in which conflicting or ambiguous needs are resolved.

An important step in identifying the operational needs for the system is to supply a rationale for each identified need. This rationale will help to ensure that both the customer and the systems engineers have a clear understanding of the need and its importance to the system (Asbjornsen 1990).

The operational needs form a basis for defining the system concept, system requirements and, eventually, the system design. Typically, only a low level of resources and time are allocated to this phase. As such, formulations about the system concept or design made during this phase are at a high level and require much further analysis (Asbjornsen 1990b). The result of the operational needs phase is an Operational Needs Document in which answers to the above-listed questions and any assumptions are clearly documented. This document should provide a clear statement of the customer's needs that is satisfactory to both the customer and the systems engineers.

The following sections in this chapter discuss the steps involved in determining the operational needs for the XMISE project and the review of the XMISE Operational Needs Document. Sample operational needs from the XMISE Operational Needs Document are provided. The chapter ends with a discussion of the lessons learned during this phase in the Systems Engineering Life Cycle.

Identifying System Users

An important activity during the operational needs phase is identifying and understanding the proposed users of the system. A cardinal rule in human factors is

to know, understand, and respect the user (Kantowitz and Sorkin 1983). The most expensive system errors are caused by not understanding the user (Mayrhauser 1990). Users include not only personnel who will physically use the system, but also those who will maintain the system. Identifying the users of the system at the beginning of the project enables system designers and developers to consider and incorporate the users' skill levels and goals into all phases of the system development process. Incorporating the user into the development process will increase the potential for a system that users may effectively and efficiently use and enjoy.

For the XMISE project, three user groups were identified. The first group consisted of skilled software developers who were experienced in writing software applications; had a Bachelor of Science in computer science, math, or electrical engineering; and had some training in the X Window System, OSF/Motif, and the UIMS. The users in this group were expected to use XMISE to lay out a user interface to an application, specify the behavior of the objects in the user interface, and add hooks to the application itself. The second group consisted of inexperienced software personnel. These users would employ XMISE to simply lay out the screens in a user interface and, as such, would not need experience in computer science or writing software. The third group consisted of maintenance personnel, who are typically overlooked when systems are developed (Baker 1990). These users required the most skills, including detailed knowledge of the UIMS, programming experience with C and OSF/Motif, and knowledge of the ULTRIX operating system.

In addition to identifying the users of the XMISE system, identification of the training and documentation they would require to use or maintain the system was also necessary.

Sample Operational Needs and Rationale

To determine the operational needs of the XMISE project, several meetings were held with the customers, potential users, and the department manager. Determining the needs and accompanying rationales as a team ensured that different system perspectives were discussed and understood by all team members. The operational needs for the XMISE project were categorized into functional needs, performance needs, operational needs, environmental needs, maintenance needs, and constraints. Below is an example of a documented need and rationale from each of the above categories.

Functional Need:

> XMISE must provide the standard OSF/Motif components for building an interactive graphical user interface. These components include objects such as windows, dialogs, pull-down menus, pop-up menus, scroll bars, list boxes, radio buttons, pushbuttons, etc. COMSAT has purchased OSF/Motif due to its functionality, flexibility, and adherence to standards, and wants software developers to use it for developing interactive user interfaces. Users recognize the components and understand how to use them in an interactive user interface.

Performance Need:

> XMISE must provide feedback to a user's requests and commands such that the user does not become distracted, confused, or impatient. The working conditions and efficiency of the users are adversely affected by distractions, confusion, or impatience. This adversely affects the competitive position of COMSAT.

Operational Need:

XMISE must make use of and operate with the UIMS Toolkit. COMSAT has spent more than two person years of effort and money on developing the UIMS Toolkit. XMISE will demonstrate the feasibility of using the toolkit and will allow other software developers to use it in their applications.

Environmental Need:

The interface to XMISE must be adjustable for a variety of lighting conditions and display hardware resolution. Improper or inadequate lighting conditions or poor resolution of the workstation may adversely affect the user's ability to interpret information and may lead to higher frequency of errors. This leads to less efficiency and lower productivity.

Maintenance Need:

XMISE must be written using a standard programming language and style. In addition, the XMISE design must be supported by a standard methodology. Standard programming language and style reduce confusion and enhance the maintainability of the software. COMSAT has determined that following a standard design methodology also makes the software easier to maintain.

Constraint:

XMISE must be developed using OSF/Motif, the X Window System, and the UIMS Toolkit. COMSAT has purchased the first two packages due to their functionality, flexibility, and adherence to standards, and wants software developers to use them. The UIMS Toolkit contains convenience routines that will help with developing XMISE.

Operational Needs Review

An Operational Needs Review Meeting was held after the completion of the draft XMISE Operational Needs Document. This meeting was attended by the customers, users, reviewers, developers, the department manager, and the advisors from both the University of Maryland and Hood College. The review provided a framework in which all participants could examine the needs of the system and agree on their importance and correctness. In addition, missing needs were identified and ambiguous needs were clarified to everyone's satisfaction. Suggestions made during the review were incorporated into the final version of the XMISE Operational Needs Document, which then became the baseline specification for the XMISE project.

Lessons Learned

Determining operational needs for a system is difficult. Different users with different perspectives sometimes do not agree on the importance or appropriateness of a need. In addition, customers and users change their minds. The iterative nature of determining operational needs cannot be overemphasized. Stating the rationale behind a need was found to be invaluable because it clarified the importance of the need for both the developers and customers. The rationale also provided a means for tracing a particular system function or feature back to a customer need.

During the first iteration of defining the operational needs for XMISE, the detailed description of the user was omitted. This description revealed additional system needs that later had an impact on both the definition of requirements and the design of the user interface. Because the user description also identified training

needs, COMSAT was able to use this information to plan and budget for user training.

CHAPTER 2. DEVELOPING THE SYSTEM CONCEPT

After determining and analyzing the operational needs of a system, the feasibility of the system must also be analyzed. A system may be infeasible because it requires technology or resources which are unavailable (Blanchard and Fabrycky 1981). The feasibility analysis explores and evaluates alternative solutions that meet the needs of the system. Ideally, trade-off studies should be performed to determine the solution that meets the needs of the system within the identified constraints (Asbjornsen 1988). The completed feasibility study recommends the preferred system concept.

In the case of the XMISE project, COMSAT was committed to keeping up with state-of-the-art user interface development standards. The X Window System and OSF/Motif had been purchased and the development of the UIMS had been under way for 2 years. In order to stay competitive, COMSAT needed to develop and demonstrate high-quality, interactive user interfaces to its software applications. Even though COMSAT needed a system to help build these interactive user interfaces, it was concerned about the feasibility of the XMISE project. Before committing to the development of XMISE, COMSAT wanted to know the answers to the following questions:

- Are resources, manpower, hardware, and software available to perform the project?
- Is funding available to support the project?

- Does a commercial, off-the-shelf package exist that meets the needs of the system and, if so, does it cost less than developing an in-house product?

Each question was addressed separately.

The XMISE developers quickly determined that COMSAT had the hardware and software resources available to design, document, and implement the system. In addition, a working prototype of XMISE existed and was available for whatever purposes the XMISE team deemed necessary. Personnel were available since the graduate students were performing the phases of the system life cycle partially at work, but mostly on their own time.

In order to justify funding through the end of 1990, the benefits of the XMISE project to COMSAT were identified. As an example, one of the benefits stated that XMISE would ensure that user interfaces developed at COMSAT have a similar look and feel, be of consistent high quality, and require less time to develop than with other available means. Funding was secured for 3 months.

After determining that the XMISE project was feasible in terms of resources and funding, alternatives to developing an in-house system were examined. A few software packages based on the X Window System and OSF/Motif such as The Builder Xcessory, XBuild, ezX, and TeleUse began appearing on the market during the summer of 1990. These packages allowed a user to graphically build a user interface and generate UIL code. These packages were evaluated for compliance with the operational needs. The evaluation consisted of a literature review of the package, attending a demonstration of the package, and/or evaluating a demonstration copy of the package in-house.

Because these software packages were new, they did not perform all of the functions required by the customers. In some cases, the user interface to the package was inadequate, cumbersome, or confusing. Because COMSAT required the system to be available to several users simultaneously, it had to purchase an expensive site license. The greatest disadvantages of these packages were:

- They would not easily interface with the existing UIMS.
- As COMSAT received new releases of the X Window System and OSF/Motif, it would be unable to use these releases until the purchased package was updated to incorporate the features of new releases.

After evaluating commercially available software packages, the concept of developing an in-house, graphical WYSIWYG user interface editor was re-examined. The advantages and disadvantages of developing a COMSAT product were discussed. For example, by developing its own product, COMSAT could customize the product for its own needs and standards and include it as one of the UIMS tools. In addition, while developing and maintaining the product, COMSAT would gain much-needed expertise in the X Window System, OSF/Motif, and user interface design techniques and standards. As the tool was implemented, it would provide an extensive test case for the UIMS toolkit. Furthermore, COMSAT had already invested 2 years of research and development in the UIMS and the XMISE prototype. The greatest disadvantage of developing an in-house product was that resources would need to be available for maintaining and upgrading XMISE as new releases of the X Window System and OSF/Motif and new user interface techniques emerged.

After examining the advantages and disadvantages of developing an in-house product, and comparing development costs with the purchase costs of an available package, COMSAT decided to pursue developing the XMISE project.

The output of this phase should be a completed feasibility study, which should identify the various system concepts explored and document the results of any trade-off analyses performed. The study results in a preferred system concept based on cost, time, and technical merit.

CHAPTER 3. THE PROJECT MANAGEMENT PLAN

Before designing and building the chosen system concept, a plan must be defined for managing the development of the product. "Systems engineering management involves planning, organizing, staffing, monitoring, and controlling the process of designing, developing, and producing a system that will meet a stated need in an effective and efficient manner" (Blanchard and Fabrycky 1981). Recognizing the expertise and skills required to perform each phase in the system's life cycle, and creating a well-integrated technical team that has these skills and expertise is an important part of systems engineering management. By teaming experts from differing engineering disciplines, the engineering specialities of individual team members may be incorporated into the system design (Asbjornsen 1988; Blanchard and Fabrycky 1981). A well-integrated team is capable of focusing on a common goal, wherein team members consider all disciplines rather than individually focusing on their own component of the system to the exclusion of the other disciplines. The team approach allows members to understand their role within the entire system and the interaction of their speciality with other engineering disciplines.

An engineering management plan documents the system development process; identifies the major project tasks, and the deliverables of each task; indicates the reviews to be held; and provides a schedule and cost estimate for the defined tasks. It also names the personnel assigned to perform each task. Depending on the customer and the size and complexity of the system, a systems engineering management plan can be quite detailed and may require strict adherence to guidelines such as those for the Department of Defense. Military Standard 499A, for example, details the

structure of systems engineering management and describes what the management plan should contain (Eisner 1988).

Project Planning and Control

Project planning may begin with the creation of a work breakdown structure (WBS) (Eisner 1988). A WBS is a hierarchical structure that separates work into logically related units (Blanchard and Fabrycky 1981). Each unit may be further decomposed into work packages that consist of a set of logically related tasks. Each work package is assigned to an organization, group, department, or team that is responsible for performing the tasks within that package.

Once the WBS is established, a cost breakdown structure (CBS) may be created. A cost estimate is made for each work package in the WBS and is based on the tasks to be performed, the number and type of resources required to complete each task, and the amount of time required to complete each task (Asbjornsen 1990c).

The WBS and CBS do not provide project timing or sequencing information. This is provided by a project schedule, which captures not only timing and sequencing information, but also the interdependent and concurrent relationships of project activities. The schedule may consist of several levels. The highest level will include each major WBS element, and the next lower level will contain the schedule for the work packages within each WBS element. At the lowest level, each work package will have a schedule showing the relationships between tasks within that work package. The schedules at each level should include major project activities, events, and milestones.

The project schedule may be represented by a PERT chart (Eisner 1988), which will show the interdependent and independent relationships between different project activities as well as the critical path. The critical path "is a chain of activities that must start and end on time for the schedule to be met" (Mills, Linger, and Hevner 1986). If any activity on the critical path is delayed, the entire project schedule will be delayed. Thus, activities on the critical path must be closely monitored by project management.

The PERT chart may also be used for management control. At certain stages during the Systems Engineering Life Cycle, performance evaluations are conducted to determine schedule and cost variances. If a task on the critical path is behind schedule or over budget, resources may be reallocated from tasks that are not on the critical path or that are under budget. The PERT chart must be updated with actual start and finish dates of tasks throughout the life cycle. From these dates a new critical path may be determined, allowing project management to better control the project.

Throughout the project, project management must assess budget, schedule, and technical risks in relation to satisfying the system requirements and specifications (Eisner 1988). A budget risk refers to the chance that the project may overrun its budget. A schedule risk refers to the possibility that the project may not be completed on time. Technical risks are project-dependent and may be the most complex to identify and assess (Eisner 1988). Risk analysis should be performed in all three areas to determine the steps necessary to control and minimize risks. Program managers should inform the customer of project risks as soon as they are known, and provide a plan for controlling and resolving the risks.

The XMISE Management Plan

The project management plan for XMISE identified the project deliverables, presented the schedule, discussed the allocation of project resources, and projected labor costs for the project. In addition, it discussed the configuration management scheme that would be used to maintain documentation, source code, and test cases.

The XMISE WBS was based on the Systems Engineering Life Cycle. Time, money, and resources were allocated to each phase in the life cycle based on the tasks that needed to be performed.

The schedule for the XMISE project was developed using MacProject II a project management tool available for the Macintosh computer. Using MacProject II (MacProject II), a PERT chart was produced of the tasks to be completed during the development of the project. It highlights the critical path and allows personnel resources to be allocated to each task. Based on the initial project start date and the duration of each task, MacProject II (MacProject II) computes the earliest start date for each task.

The initial, unrealistic XMISE schedule shown in Figure 2 includes the start date for each phase of the XMISE development and the scheduled date for the review following each phase. It also identifies separate documentation and prototyping activities. The critical path is represented by the bold black line. The activities on this path have the potential to delay the entire project.

Using the software, a table of personnel resources was built which included the work week schedule for each person and their fully loaded cost per week. Personnel resources included the developers, reviewers, customers, the department manager,

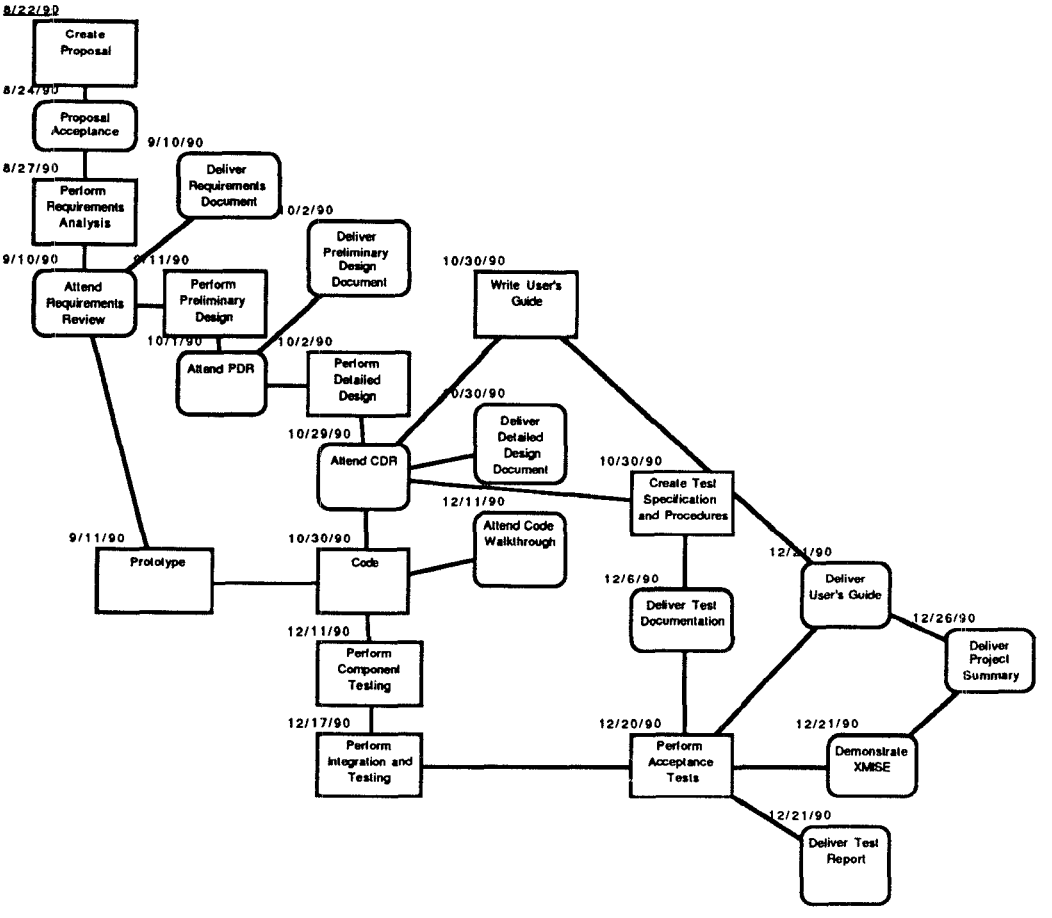


Figure 2. The Original XMISE Schedule

and the two academic advisors. These resources were then allocated to the phases requiring their expertise. No other costs were incorporated since the labor costs included company overhead, and COMSAT already had the software and hardware necessary for implementing the XMISE project. From this resource allocation, costs were computed for all personnel except the academic advisors. Using these figures, MacProject II (MacProject II) tracks the total project cost based on the resource allocation and the specified schedule, and displays the costs in a cash flow table

showing cumulative costs. The schedule and cost estimates did not include preparing training materials or training users.

The XMISE Project Management Plan was updated throughout the development of the project to reflect milestone completions, and schedule and cost deviations. As a result of schedule and cost deviations, it included re-planning and re-scheduling. By evaluating these deviations, schedule and budget risks were identified. The solutions for controlling these risks had an impact on the development effort as described in Chapter 7.

Together with the schedule and costs, the XMISE Project Management Plan included the configuration management scheme for the XMISE documentation, source code, and test cases. A project support environment was established on the Macintosh for the XMISE documentation using COMSAT's internal project support environment, SCOPE. All documentation, including documents, schedules, costs, correspondence, action items, discrepancies, and presentations, was controlled by SCOPE and produced on the Macintosh using available Macintosh tools such as Microsoft Word (Microsoft Word Version 4.0), Microsoft Excel (Microsoft Excel Version 1.5), ANATOOL (ANATOOL), MacDraw II (MacDraw II Version 1.1), and MacProject II (MacProject II).

In order to control the source code, separate directories were established on the ULTRIX system for source code development, configuration control, the current baseline version of the executable software, and test cases. Procedures were established for creating a new baseline version of the executable.

Lessons Learned

Scheduling system tasks and estimating costs require experience and a realistic idea of the time required to complete a task. Schedules and cost estimates must include time and budget for incorporating feedback into the result of a phase, as well as iteration within a phase. Feedback includes suggestions made during review meetings, as well as errors of omission or commission discovered during the performance of the phase or during a subsequent phase.

One of the most important lessons learned is that identifying, analyzing, and managing project risks cannot be overlooked. During development of the XMISE project, risks were not clearly identified or understood. If they had been, the schedule and budget could have been adjusted accordingly. Barry Boehm identifies, and provides management techniques for, the following top 10 software risk items (Boehm 1988):

- Personnel shortfalls
- Unrealistic schedules and budgets
- Developing the wrong software functions
- Developing the wrong user interface
- Gold-plating
- Continuing stream of requirement changes
- Shortfalls in externally furnished components
- Shortfalls in externally performed tasks
- Real-time performance shortfalls
- Straining computer science capabilities.

By acknowledging that risks exist, a plan can be established for resolving and managing each risk item. An examination of the personnel resources for managing and developing XMISE would have revealed that schedule and budget risks existed because the developers would be working with new software (UIMS, OSF/Motif, the X Window System, and ULTRIX), as well as performing the entire system life cycle, for the first time.

CHAPTER 4. SYSTEM REQUIREMENTS DEFINITION AND ANALYSIS

The contents of the Operational Needs Document provide the basis for deriving the system requirements, which establish the necessary features and functions of the system that will meet the stated operational needs (Asbjornsen 1990b) and provide a clear picture of what a viable solution entails (Mayrhauser 1990). Defining and analyzing the system requirements is the fourth phase in the Systems Engineering Life Cycle as shown in Figure 1. This phase typically requires and is allocated more time and resources than the operational needs phase and, as such, provides a much more detailed, less abstract basis for developing the system concept and design. The results of the requirements definition and analysis phase are documented in the System Requirements Document, which includes a description of the system's environment and presents the system's functional, operational, physical, maintenance, interface, and future requirements, as well as system constraints. Each requirement has an associated rationale similar to an operational need. This document should state the requirements and associated rationales in a clear, unambiguous way using the customer's terminology. These requirements must then be approved by the customer.

The following sections describe the steps in defining and analyzing requirements in general and for the XMISE project. The System Requirements Review is also discussed. Sample requirements from the the XMISE Requirements Document are provided. The chapter ends with a discussion of the lessons learned while performing this phase in the Systems Engineering Life Cycle.

Defining Requirements

As in the operational needs phase, customers and users play a vital role in defining requirements. Interviews and meetings with the customers and users provide a means for systems engineers to understand the customers' and users' perception of the system's function, operation, and performance. These perceptions, together with the operational needs, are translated into system requirements. By using the words shall, should, or may in a requirement, the customer conveys the relative importance of requirements (Eisner 1988).

Requirements may be grouped into the following six categories: functional, physical, operational, maintenance, interface, and future requirements (Asbjornsen 1988). Functional requirements addresses what the system does. The composition of the system in hardware, software, and humans is defined by physical requirements, which specify physical constraints on the system design and may constrain the allocation of functional requirements to hardware, software, or humans. Operational requirements defines how users and customers want to operate or use the system and include desired protection from user errors. The desired system availability, maintainability, and reliability are defined by maintenance or logistics requirements. Interface requirements defines other systems with which the system must interact, including people and the environment. Future requirements defines whether the system should be capable of evolving or changing with changing user needs, available technology, cost constraints, or the environment. Each of these categories typically includes performance requirements.

Sample Requirements

Defining the requirements for the XMISE project involved analyzing the operational needs in the XMISE Operational Needs Document; meeting with customers, potential users, and the department manager; examining the prototype system; and collecting users' suggestions and reactions after using the prototype. Several meetings were held to discuss whether the customer wanted certain features of the prototype to remain. Suggestions for improving the prototype were also made during these meetings and additional features and functionality were identified and documented. Detailed notes were taken during the team meetings and interviews with prototype users. From these notes, concise requirements were written.

The XMISE requirements were categorized into functional, error-handling, physical, user interface, operational, maintenance, interface, testing, and future requirements. System constraints were also stated. Because the user interface to the XMISE system was extremely important to the customers and users, the requirements for it were placed in a separate category from general interface requirements. Some of the above named-categories were further divided into subcategories, which included a set of related requirements. Below is an example of an XMISE requirement and rationale from each of the above specified-categories.

Functional Requirement:

> XMISE shall generate UIL code that defines the interactive user interface from the actions the user has performed while using XMISE. UIL code is the language that OSF/Motif reads to create a user interface. In addition, if XMISE generates UIL code, the user interface may be changed without recompiling the application.

Error-Handling Requirement:

Users shall not be allowed to perform illegal actions. If the user tries to perform an illegal action, nothing should happen. Being allowed to perform an illegal action may frustrate and confuse the user, resulting in reduced productivity.

Physical Requirement:

The implementation of XMISE shall be independent of the display hardware platform. Because XMISE must run on several hardware platforms, it cannot incorporate a specific feature of the hardware platform on which it was implemented.

User Interface Requirement:

XMISE shall be an interactive menu- and dialog-driven system. COMSAT users prefer interactive user interfaces and find menu and dialog systems easier to use than command-driven systems.

Operational Requirement:

XMISE shall complete system initialization within 20 seconds. Users may become impatient or irritated if the system takes too long to initialize, leading to a loss of productivity.

Maintenance Requirement:

XMISE shall be kept compatible with new releases of the X Window System, OSF/Motif, and the UIMS toolkit. If XMISE is not updated, it will quickly become useless since all other software will be developed using the updated releases of these systems.

Interface Requirement:

XMISE shall produce UIL files in the OSF/Motif user interface language format. OSF/Motif understands a UIL file only in its own format.

Testing Requirement:

The XMISE documentation shall include a Test Specification, a Test Procedures Document, and a Final Test Report. These documents are necessary to demonstrate that XMISE works as envisioned.

Future Requirement:

XMISE should run under the VMS operating system, supporting the X Window System and OSF/Motif. COMSAT also develops applications under the VMS operating system and would eventually like to create interactive user interfaces that work in this environment.

Constraint:

The features and functions of XMISE are constrained to those supported by the X Window System, OSF/Motif, and the UIMS toolkit. COMSAT does not want to invest time or money in additional software, as it has these packages available.

Requirements Analysis

Analyzing the requirements of the system provides a first-level decomposition of the system into subsystems or components that provide well-defined, manageable functions (Asbjornsen 1988). This analysis is usually presented through diagrams that illustrate the major components of the system and the information flow and/or control flow between the subsystems and the external environment. These diagrams

may be functional requirements diagrams (Mayrhauser 1990), functional block diagrams (Blanchard and Fabrycky 1981), data flow diagrams, or box structure diagrams (Odom 1990). This top-down analysis is repeated on each individual subsystem, refining it into more and more detail until the functions of the system and their interactions are well-understood.

Figure 3, a context diagram of the XMISE system, illustrates the external entities that interact with the system. Diagrams were created for the first several levels of system decomposition using ANATOOL (ANATOOL), a CASE tool that supports the Yourdon methodology of Structured Systems Analysis. It allows the user to generate data flow diagrams using the conventions of either Yourdon or Gane and Sarson. The tool also produces a data dictionary.

Requirements analysis also involves allocating functions to hardware, software, and persons, or a combination of these three. Certain functions are better performed by humans than computers, and certain ones are better performed by computers than humans. For example, humans excel in reacting to unexpected events, where as computers excel in performing routine, repetitive, or very precise operations (Baker 1990). Using known human factors guidelines and the system constraints, each system function is examined to determine if it should be realized through hardware, software, persons or a combination, and is allocated accordingly.

Requirements Traceability Matrix

An important step in requirements definition and analysis is establishing a means by which requirements may be traced through the system design and implementation. Typically, the customer wants some proof that each requirement was satisfied (Eisner 1988). The first step is to list the requirements and assign each

one a unique number. Specific CASE tools exist to support requirements traceability, but a spreadsheet works quite well. As the system is decomposed into functional subsystems or components, the requirements are allocated to these components. The list of requirements is updated to reflect which component will satisfy each requirement.

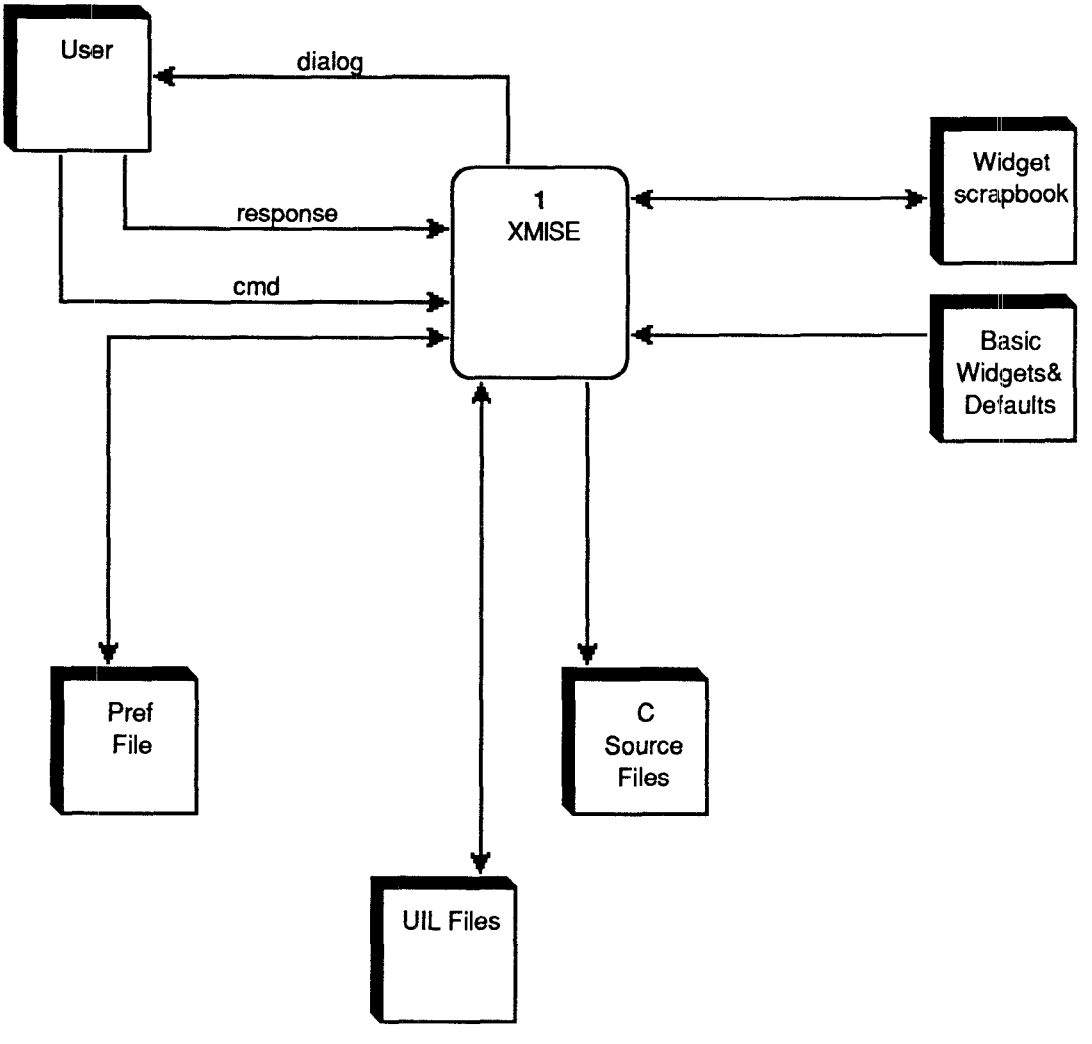


Figure 3. The XMISE Context Diagram

A requirements traceability matrix was created for XMISE using a Microsoft EXCEL (Microsoft EXCEL Version 1.5) spreadsheet. This spreadsheet was updated throughout the system's life cycle, and tracked where each requirement was satisfied in the design, implementation, and testing phases.

System Requirements Review

A System Requirements Review Meeting was held after the draft of the XMISE Requirements Document was completed. This meeting was attended by the customers, users, reviewers, developers, the department manager, and the advisors from both the University of Maryland and Hood College. The review provided a framework in which all participants could examine the requirements, as well as the functional decomposition, of the system. This was the most critical review meeting, because it enabled the customers to determine whether the developers of the system understood the systems requirements and whether all of the operational needs had been addressed. The customers could also correct any misunderstandings and identify any missing requirements before the system design began. Suggestions made during the review were incorporated into the final version of the XMISE Requirements Document, which then became the baseline from which the system would be designed.

Lessons Learned

Like determining operational needs for a system, defining the requirements for a system is difficult. Different users have different requirements and do not necessarily agree on the relative importance of a single requirement. In addition, customers and users change their minds. The iterative nature of requirements definition and analysis

cannot be overemphasized. In fact, the iterative nature of the process begins to resemble an infinite loop in a computer program. Iterating through requirements can become a trap that eats budget and time. At some point, a collective decision must be made to freeze the requirements so that the system may be designed and implemented.

CHAPTER 5. DESIGNING THE USER INTERFACE

The user interface to a system is that portion of the system that interacts with the user. For example, the system could be an automobile or a computer program. Each of these systems has a required set of user actions and procedures that makes the system work. Designing a user interface necessitates that human factors requirements for the system be well-understood in addition to understanding who the users are, their skill levels, backgrounds, and the goals they want to accomplish with the system. The Human Factors Society defines human factors as "the discovery and application of principles concerning human behavior and characteristics to the design, evaluation, operation, and maintenance of products and systems that are intended for safe, effective, satisfying use by people." The goal of human factors is to optimize the relationship between human beings and systems (Kantowitz and Sorkin 1983). In order to optimize this relationship and ensure that the system is safe, effective, and satisfying, human factors must be properly integrated into the system life cycle phases along with hardware, software, and other system components (Blanchard and Fabrycky 1981). As technology progresses, systems are becoming increasingly complex and costly. Omitting consideration of the human being as integral to the system leads to an ineffective, inefficient system, one that is costly to retrofit (McCormick and Sanders 1982). Human factors requirements are derived from the operational and maintenance requirements of the system (Blanchard and Fabrycky 1981).

Designing a user interface also requires applying and understanding known human factors guidelines and data for the type of user interaction required by the

system. And, it requires experience to know when and where to apply these guidelines (Rubinstein and Hersh 1984).

Designing a Human Computer Interface

Designing human computer interfaces that are "simple but powerful, easy to learn but appealing to experienced users, and facilitate error handling but allow freedom of expression" (Shneiderman 1980) is a challenging task. However, if designers take time to plan the interactive user interface, understand the users' needs, pay close attention to detail in design and development, and take time to thoroughly test the system, a high-quality interactive user interface can result (Shneiderman 1987). Rubinstein and Hersh believe that designers with experience, intuition, a knowledge of human factors guidelines and rules, and exposure to user behavior theory can produce a good user interface design (Rubinstein and Hersh 1984).

Several sources of guidelines exist for human computer interfaces. Rubinstein and Hersh present more than 93 general guidelines such as "verify that help helps" and "keep menus short" (Rubinstein and Hersh 1984). Shneiderman provides specific guidelines for different interaction styles, including eight rules for dialog design (Shneiderman 1987). The U. S. Government also has several handbooks and guidelines for human computer interaction, including 944 guidelines that address data entry, data display, sequence control, user guidance, data transmission, and data protection in *Guidelines for Designing User Interface Software* (Smith and Mosier 1986).

These guidelines range from very general to very specific. As an example, the Open Software Foundation suggests the following seven guidelines for designing user interfaces that are consistent and easy to use (Open Software Foundation 1991):

1. Adopt the user's perspective.
2. Give the user control.
3. Use real-world metaphors.
4. Keep interfaces natural.
5. Keep interfaces consistent.
6. Communicate application actions to the user.
7. Avoid common design pitfalls.

These guidelines and rules may be applied to software systems with different interaction styles. Software systems may support user-interaction through menus, command languages, direct manipulation, or a combination of these (Shneiderman 1980; 1987; Mayrhauser 1990).

Designing the XMISE User Interface

The XMISE user interface design team consisted of experienced designers knowledgeable in human factors guidelines for designing interactive human computer interfaces. In addition, the requirements were well-defined and could be used to guide the design (Rubinstein and Hersh 1984). The XMISE system required the user interface to be an interactive menu- and dialog-driven system, implemented using OSF/Motif, the X Window System, and the UIMS toolkit. This requirement implied that the interaction style of the user interface would be a combination of menus and direct manipulation; provided a focus for the design; and identified the types of user interface guidelines to follow. In addition to experience, two primary sources were used as guidelines for designing the user interface to XMISE. The first was the OSF/Motif Style Guide, and the second was the actual user interface conventions used by most Macintosh-based applications. The OSF/Motif guidelines were chosen

because the system was being developed with OSF/Motif and its style guide provided general guidelines as well as detailed rules of thumb for choosing the appropriate interface object for a particular user action. The selection to try following Macintosh conventions and styles was based on the fact that the prospective users of the system were proficient Macintosh users. If the XMISE user interface could employ conventions similar to those found in a typical Macintosh user interface, users could transfer their knowledge gained from using Macintosh user interfaces to using the XMISE user interface.

In addition to applying human factors guidelines and meeting the XMISE requirements, the design of the XMISE user interface involved evaluating the user interface of the working prototype. This evaluation included discovering tasks that were difficult for a user to perform, identifying missing functions, and noting where the interface was awkward to use. It also included incorporating feedback from users who had used the prototype.

The XMISE screens were designed using the in-house UIMS MacDraw II (MacDraw II Version 1.1) library for Motif widgets. This tool allows the designer to draw screens that look very similar to how they will look when implemented using OSF/Motif and the UIMS. All of the dialogs with which the user would interact while using XMISE were created. These user interface screens differed substantially from the existing prototype screens.

The XMISE Screen Specification consisted of the XMISE screens, together with an explanation of the function of each screen and the individual objects on the screen. This explanation was written from the user's perspective such that the user

could envision how to use a particular dialog to build a user interface. Figures 4, 5, and 6 present sample XMISE user interface screens.

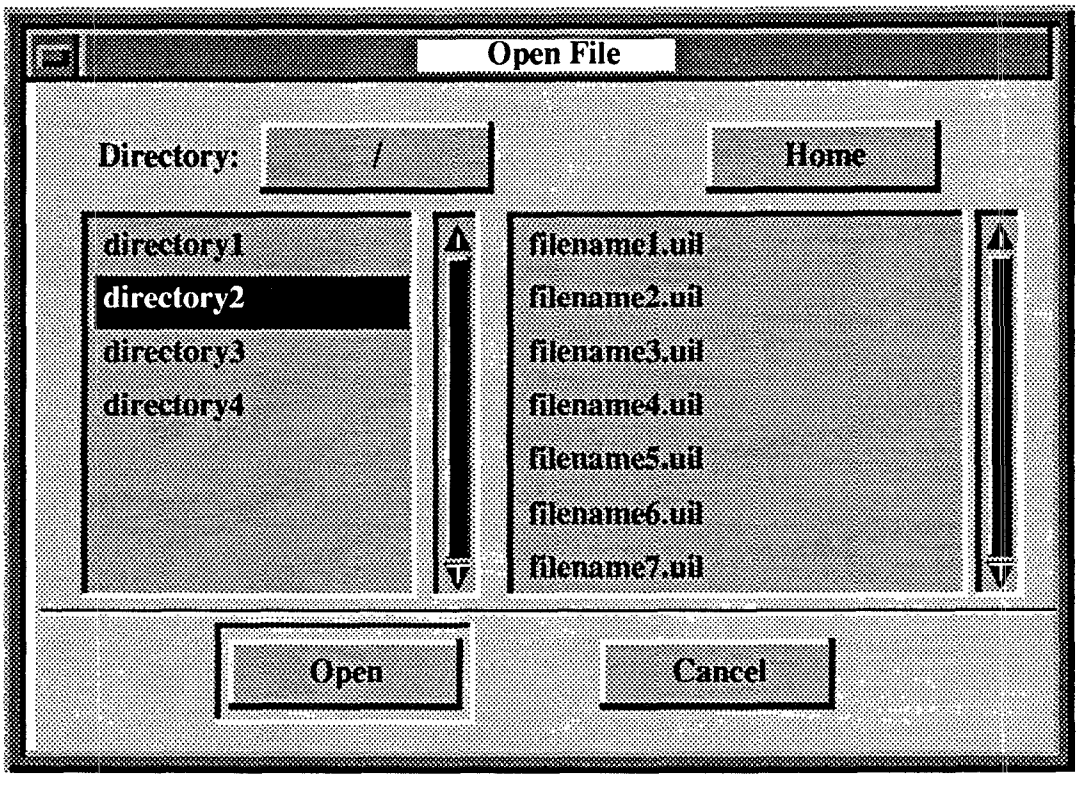


Figure 4. Sample XMISE Dialog for Opening a File

Figure 4 illustrates the dialog that allows a user to open a previously created UIL file using XMISE. The dialog includes features that allow the user to traverse an ULTRIX directory structure, quickly return to the directory from which the XMISE application was invoked, and open a file by double-clicking on a filename, or single-clicking on a filename and clicking the Open button.

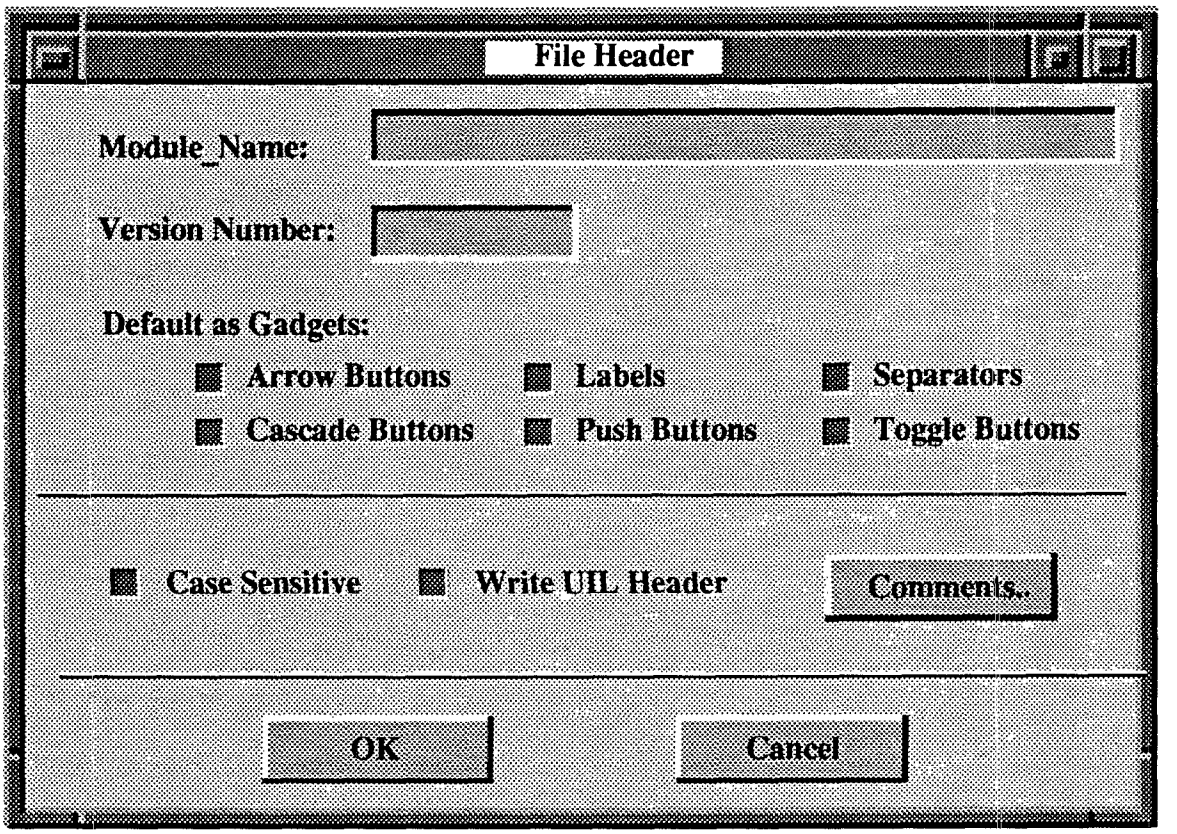


Figure 5. Sample XMISE Dialog for Specifying UIL File Characteristics

Figure 5 illustrates the File Header Dialog, which allows a user to specify header information for the UIL file being created or modified by XMISE. It provides a means for the user to set defaults that will be applied throughout the UIL file and includes a feature that allows a user to enter a free text comment that will be included in the top of the UIL file. This option allows the user to specify, for example, a file prologue.

The Values Dialog, illustrated in Figure 6, allows a user to define constants that may be used while building the user interface to an application. The user may create new values, edit and replace existing values, or delete existing values. In addition,

the dialog provides the user with feedback on the number of times a selected value is referenced by the user interface. A value that does not have a reference number of zero may not be deleted.

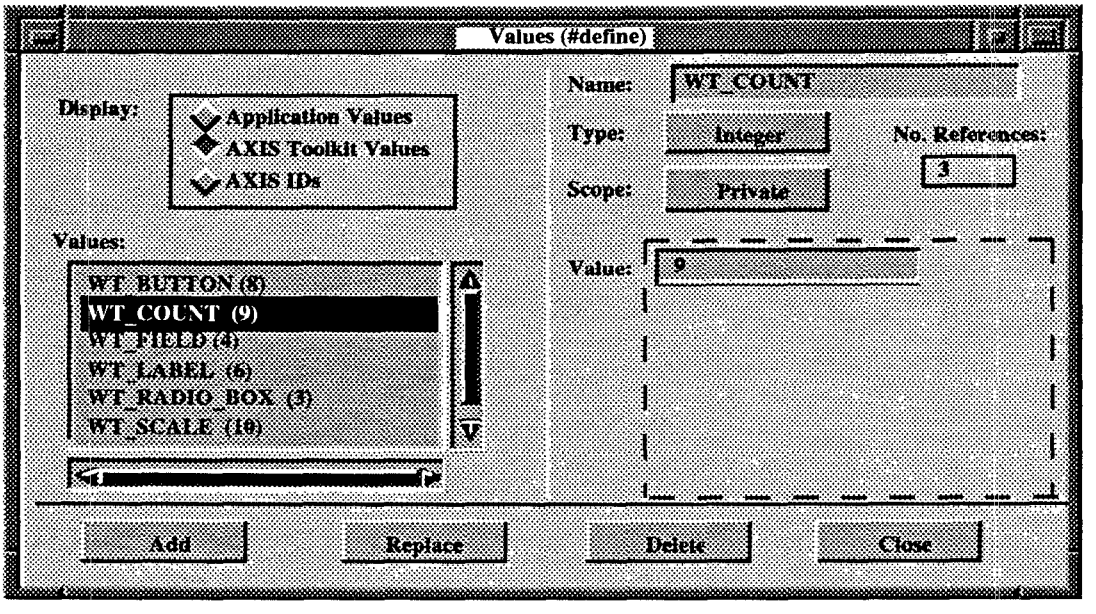


Figure 6. Sample XMISE Dialog for Specifying Values

The User Interface Review Process

A sure way to demonstrate to customers and users that the system requirements were understood is to show them a prototype or mock-up of the user interface to the system. The XMISE Screen Specification was delivered to the team members for review. Many of the screens had been reviewed iteratively during the design; however, this document included all of the screens, as well as a description of how the user would interact with the screens to produce a user interface for their application. Rubinstein and Hersh state that "design is seldom an orderly or linear

process" (Rubinstein and Hersh 1984). This was certainly true for the design of the XMISE screens. The user interface reviews that were held resulted not only in changing screen formats, but also in changing, adding, and eliminating requirements for the system. Therefore, several versions of the screens had to be produced, documented, and again reviewed, plus the requirements document had to be updated to reflect the changes.

The Requirements Traceability Matrix

Once the requirements were again agreed upon and the screen designs were decided upon, the Requirements Traceability Matrix was updated to include the screen format that implemented each functional requirement. This enabled the customer and the XMISE team to verify that each requirement had been included in the design of the system. Figure 7 illustrates a portion of the XMISE Requirements Traceability Matrix.

Requirement Description	Shall	Should	May	XMISE Component
4.2.2.2.1 Allow user to declare values and use them	X			Values Dialog
4.2.2.2.2 Allow user to modify values	X			Values Dialog
4.2.2.2.3 Read values from UIL	X			Open Option of File Menu
4.2.2.2.4 Write values to UIL	X			Save Option of File Menu
4.2.2.2.5a Allow user to define the scope of a value	X			Values Dialog
4.2.2.2.5b Allow user to specify the default value of an imported value			X	Values Dialog
4.2.2.2.6 Support specified value types	X			Values Dialog

Figure 7. A Portion of the XMISE Requirements Traceability Matrix

The matrix in Figure 7 includes information on which requirements were designated as "shalls," "shoulds," and "mays" by the customer. While this provides a means to quickly determine the importance of a requirement, a significant improvement can be made to this matrix. The "shall," "should," and "may" columns should be replaced with two columns. The first would contain an indicator of the importance of the requirement. For example, each requirement could be ranked on a scale from one to five, with one indicating a very important requirement and five indicating an unimportant requirement. This exercise should be performed closely with the customer (Rombach 1991). The second column would contain an indicator of the difficulty of designing and subsequently implementing a requirement. Similarly, each requirement could be ranked on a scale from one to five, with one indicating the requirement will be very difficult to design and implement and five indicating the requirement is simple to design and implement. This exercise should be completed by the system designers and implementors (Rombach 1991). By indicating the difficulty of a requirement, potential technical, schedule, or budget risks may be identified.

These indicators of importance and difficulty will provide much-needed information as the system development progresses. If the project begins to experience budget or schedule problems, this matrix may be examined to see which system requirements are the most important to the customer yet are the easiest to design and implement. It provides a means to perform trade-off analyses on which requirements to implement. The customer may be much more satisfied if the system performs several important requirements that were easy to implement than one important requirement that was difficult to implement. The matrix may also be used

to justify to the customer why certain requirements were implemented and others were deferred (Rombach 1991).

Lessons Learned

Designing the screens and having them reviewed before implementation saved time and money. If they had been implemented before going through the iterative review process, much of the effort would have been wasted since several requirements were found to be unnecessary. On the other hand, since the screens were the first indication of what the user interface to XMISE would look like, everyone seemed to have an opinion about at least one screen. This is another phase of the system life cycle that, if not carefully monitored, can result in an infinite loop, wasting a substantial amount of time and money. Unfortunately, this is really the last phase in which the customer still has the ability to change things before they are implemented and much more expensive to change. Therefore, designers must remain flexible and patient.

Another lesson learned is that having an on-line screen prototyping tool such as the UIMS MacDraw II library is invaluable. Creating the XMISE screens with this tool allowed them to be easily updated. In addition, a true representation of the user interface screens gives users and customers an accurate perception of the user interface to their system.

CHAPTER 6. THE XMISE SOFTWARE DESIGN

The XMISE Screen Specification was used as the basis for the XMISE software design. The specification identified all of the features and functions the software system had to provide. Because the X Window System and OSF/Motif provide the low-level functionality of displaying windows and getting user actions, software to perform these functions did not have to be designed. Software applications based on the X Window System and OSF/Motif are event-driven systems, which wait for an event to occur, then respond to the event, and wait until the next event occurs (Young 1989). Events caused by user or program actions enter an event queue, and the software responds to the events in the queue. The software enters what is essentially an infinite loop that processes or ignores the events that enter the queue and waits until the next event.

Events may enter the queue through user actions such as a mouse click or a key press; thus, there is no way of knowing which events will enter the queue or in what order. In addition, events in the queue may be unrelated. This differs considerably from traditional procedure-oriented applications. The only procedural factor in an event-driven system is the response to an event in the queue. This response can be a well-defined set of procedures that process the particular event.

Because XMISE is an event-driven system, it posed certain software design difficulties. The most popular methodology used by COMSAT was based on Edward Yourdon's Structured Systems Analysis and Edward Yourdon's and Larry Constantine's Structured Design. This methodology had worked well in the past for many of the applications developed at COMSAT, but these applications were

procedural in nature and tended to be data intensive. Conversely, XMISE is not procedural in nature and requires that the behavior or control flow of the system be modeled in addition to the data flow.

Since XMISE is control flow-oriented and Structured Analysis does not model control flow or behavior, a different methodology had to be found. The real-time extensions to Structured Analysis provided by Ward-Mellor and Hatley-Pirbhai were examined. These extensions allow the behavior of the system to be modeled in conjunction with the data flow model. These real-time extensions had an advantage over a totally new methodology, since the XMISE developers and team members already knew the Structured Systems Analysis methodology and time was a critical factor. A decision was made to analyze the system using the extensions defined by Ward-Mellor.

Performing Structured Systems Analysis for XMISE

Structured Systems Analysis was performed using ANATOOL an in-house CASE tool. ANATOOL (ANATOOL) is a Macintosh-based software tool that provides an editor for drawing data flow diagrams, and a central data dictionary for storing definitions of data elements. It is easy to use but does not include the real-time extensions that were needed. Because finding and purchasing CASE tools was not part of the XMISE project, and the only other choice was to use MacDraw II (MacDraw II Version 1.1) and thus lose the data dictionary feature, it was necessary to use ANATOOL.

A decision was made to show both control flow and data flow on the data flow diagrams. In order to distinguish control flow variables from data flow variables, all control flow variables were prefixed with "cmd_" which means command. Figure 8

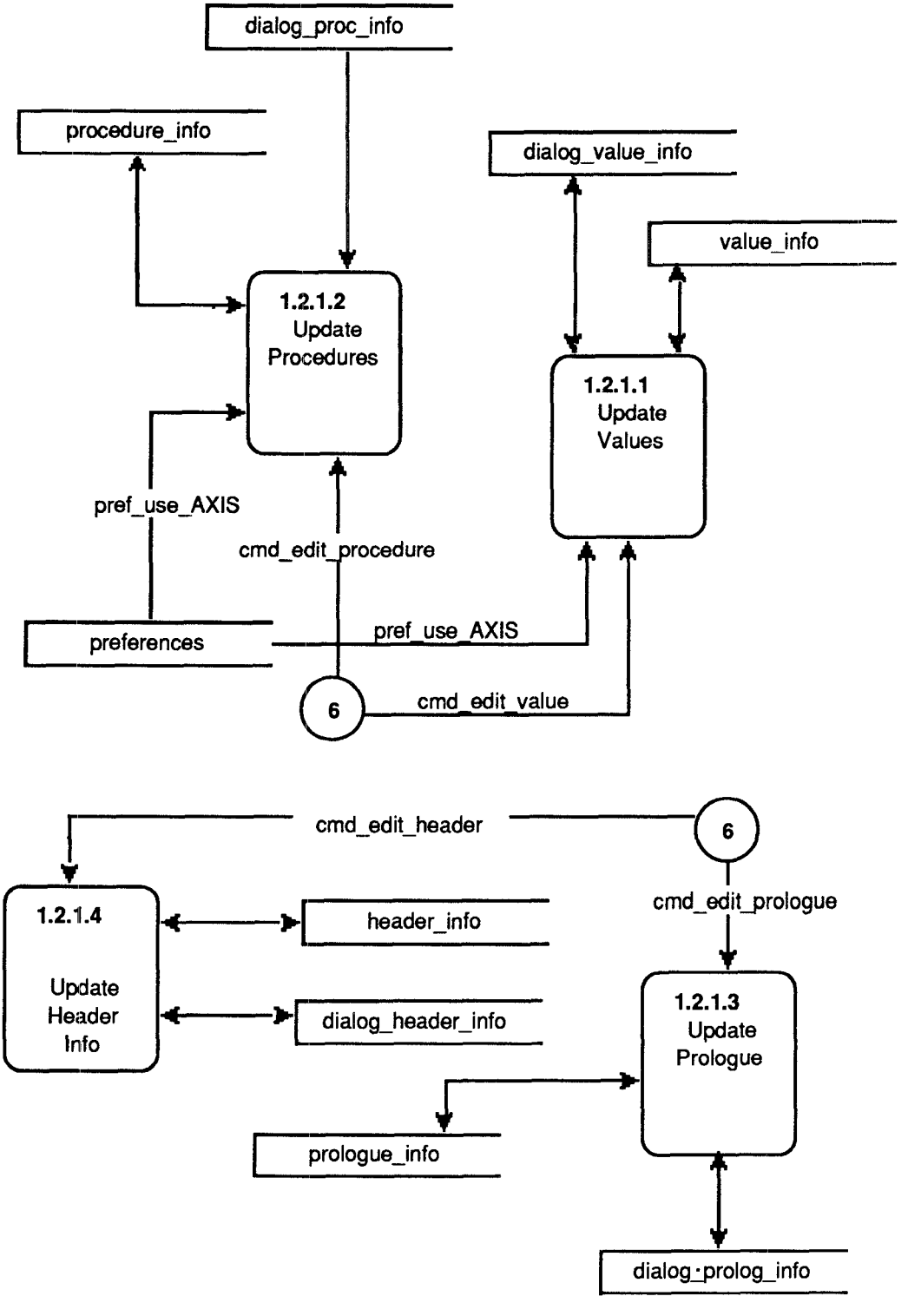


Figure 8. Sample XMISE Data Flow Diagram with Control Flow

illustrates an XMISE data flow diagram that uses this convention. For example, the process Update Values requires both data and an edit value command. Showing control flow in this manner was not ideal, but it captured both data flow and control flow on the same diagram. These diagrams did not distinguish between control transformations and data transformations.

As stated above, ANATOOL (ANATOOL) stores all elements added to a data flow diagram in a data dictionary, which allows simple and composite data flows to be defined, as well as data stores. The data dictionary is very helpful for keeping track of data items, but becomes cumbersome when trying to define and illustrate control flow variables. In order to clearly define the structure of control and the hierarchy of commands, the Warnier-Orr diagramming method was used. Figure 9 illustrates a Warnier-Orr diagram for a subset of the control variables. This diagram was created using MacDraw II (MacDraw II Version 1.1) since a CASE tool that supported Warnier-Orr diagramming was not available in-house. Because the control variables were also defined in the ANATOOL data dictionary, there was no loss of information. This diagram was found to be so helpful for the control variables, it was also used to diagram large composite data flows and data stores. The Warnier-Orr diagrams provided essential information when reading the data flow diagrams.

The analysis was completed after mini-specs were written for the lowest-level processes on the data flow diagrams. These mini-specs were documented using Microsoft Word (Microsoft Word Version 4.0).

Performing Structured Design for XMISE

The step from Structured Analysis to Structured Design requires creativity as well as care. Designing the software as a hierarchy of modules that interact with one another

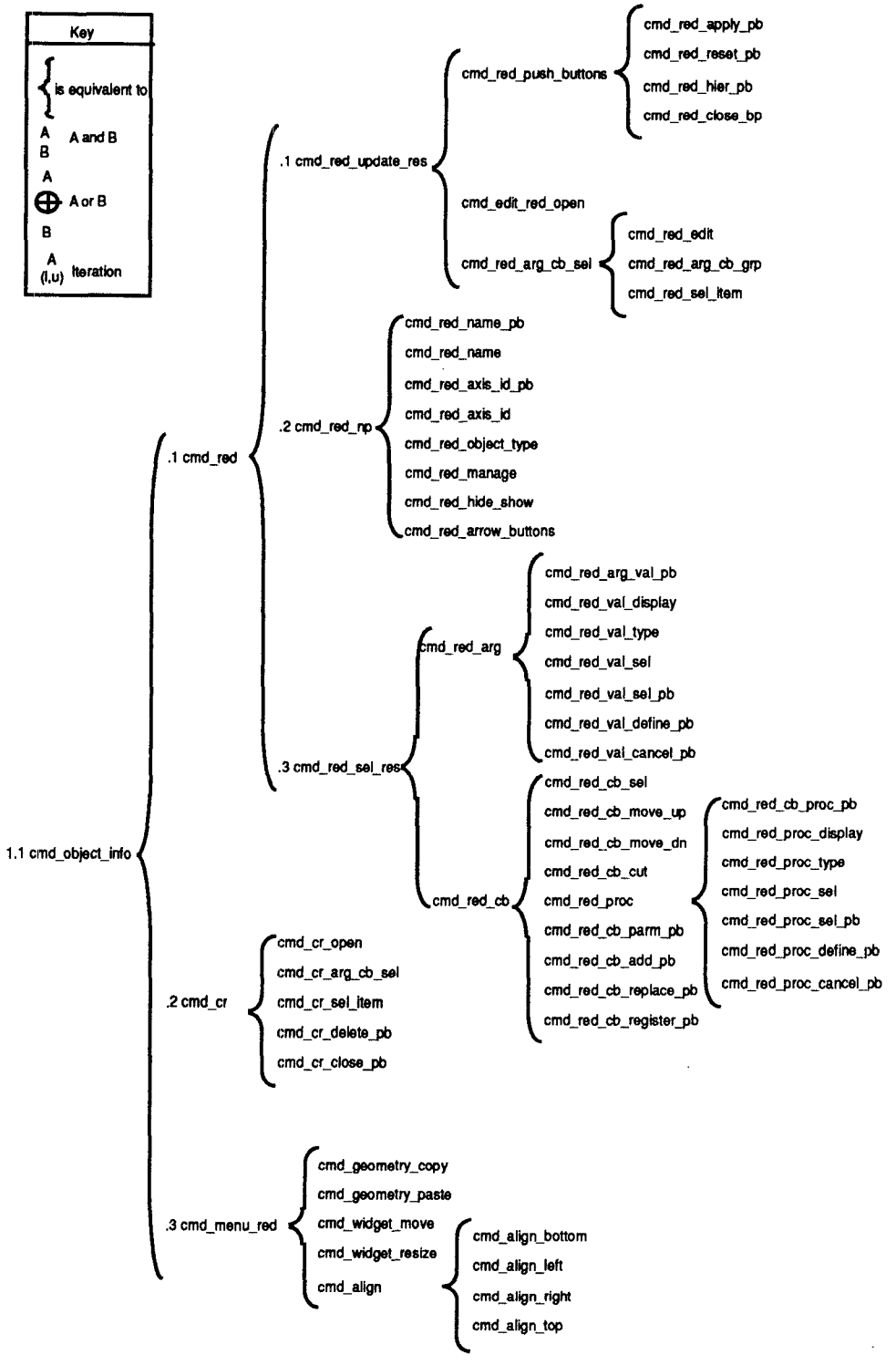


Figure 9. Sample Warnier-Orr Diagram for Control Variables

requires that all of the data and control information (when using real-time extensions) found during the analysis phase not be lost. This is typically a tedious task. In the case of an event-driven system like XMISE, creating module hierarchy diagrams did not seem appropriate for representing the design. A single structure chart or hierarchy diagram cannot be created for event-driven programs. The software does not proceed from a single point. Event-driven programs consist of several very shallow (one or two levels), unrelated module hierarchies. These hierarchy diagrams provide the processing for a single event, however, they do not model the behavior of the software. Several discussions with colleagues and Dr. John Boon at Hood College resulted in the decision that the structure charts did not provide the framework for designing the event-driven system. Therefore, structure charts were not produced as part of the design phase.

An attempt was then made to model the behavior of the system using state transition diagrams. These diagrams created even more problems than the structure charts. Deciding what should and should not be a state became a difficult and time-consuming process. An unmanageable number of states can exist in a user interface if a state is defined to be a change in the status of the user interface, i.e., a push-button label becomes insensitive. On the other hand, if states are not defined at this level, there is a risk of losing information. Time and budget constraints did not permit defining an exhaustive number of states. Thus, a decision was made that the user interface would enter a new state whenever a new dialog would appear on the screen in response to a user- or program-generated event. Any events that occurred while in a single dialog did not change the state of the system, unless a new dialog was displayed to the user.

After adopting the state definition, ICONIX's state transition diagram editor, Fast Task (Power Tools) was used to create state transition diagrams. These diagrams illustrate what actions occur in response to a given event while in a given state, and what events and actions place the system in a new state. Fast Track (Power Tools) has a nice feature that allows a diagram to immediately be displayed as a state-event matrix, a state table, or a from-state/to-state matrix. Once again, the CASE tool environment proved to be limited. The editor would allow only a single action to be associated with an event, and the diagram could not be annotated. The frustration and difficulties with the CASE tool and the unavailability of other state transition diagram editors in-house, resulted in abandoning the creation of state-transition diagrams.

The technique finally agreed on to represent the software design was to write psuedocode, using Microsoft Word (Microsoft Word Version 4.0), for each process in the data flow diagrams and describe the actions that occurred when a control variable entered the process. This technique further defined the behavior of the system, and described how the data in the system was affected by the system's behavior.

Updating the Requirements Traceability Matrix

In addition to psuedocode, correlating the screen specification to the processes on the data flow diagrams and the control variables also provided valuable design information. The Requirements Traceability Matrix was once again updated to include this correlation. Each dialog in the user interface, each object on a dialog with which the user could interact, and each action expected to be performed was entered into the matrix. Each entry was then correlated to a section in the XMISE Screen Specification, a process number in the data flow diagrams, and a control flow

variable (a variable that begins with "cmd_"). A portion of this matrix is illustrated in Figure 10. This matrix ensured that each feature and function described in the XMISE Screen Specification was included in the software design, which in turn meant that each functional requirement was also designed into the system.

The data flow diagrams, mini-specs, psuedocode, data dictionary, Warnier-Orr diagrams, and traceability matrix were combined into the XMISE Design Specification. This document also included the testing strategy for the XMISE software.

XMISE Software Design Review

A Software Design Review Meeting was held after completion of the draft XMISE Design Specification. This meeting was attended by a small portion of the XMISE team, as the users did not need to attend, and the customer needed to attend only for information purposes. The goal of the meeting was to determine if other software designers could understand the design, suggest design improvements, and thought the system could be implemented from the design. During the meeting, only a few suggestions were made concerning the design, the majority of the meeting was spent discussing testing strategies. Methods and techniques for testing interactive user interfaces appears to be a fairly new area. Because the system is event-driven, with few interrelating modules, regular testing strategies were difficult to apply. The team agreed that the testing strategy included in the design document needed to be augmented and that the XMISE Test Specification needed to be completed before implementation. The XMISE Design Document was finalized and became the baseline from which the system would be coded.

Screen Specification (SS) Action	SS Section #	DFD #	Control Flow Name
Home Dialog	3.3.1		
Widget Hierarchies List Box (SC)		1.3.2	cmd_hier_sel_top
Widget Hierarchies List Box (DC)		1.3.2	cmd_hier_hide_show_top
Show/Hide PB		1.3.2	cmd_hier_hide_show_top
Unselect PB		1.3.2	cmd_hier_unselect_top
Parent PB		1.3.2	cmd_hier_parent_pb
List Box 1 (SC)		1.3.2	cmd_hier_sel_1
List Box 1 (DC)		1.3.2	cmd_hier_dsel_1
List Box 1 (UP)		1.3.2	cmd_hier_up_1
List Box 1 (DN)		1.3.2	cmd_hier_dn_1
List Box 1 (CUT)		1.3.2	cmd_hier_cut_1
List Box 2 (SC)		1.3.2	cmd_hier_sel_2
List Box 2 (DC)		1.3.2	cmd_hier_dsel_2
List Box 2 (UP)		1.3.2	cmd_hier_up_2
List Box 2 (DN)		1.3.2	cmd_hier_dn_2
List Box 2 (CUT)		1.3.2	cmd_hier_cut_2
Name PB		1.3.2	cmd_hier_name_pb
Name Text Field		1.3.2	cmd_hier_name
AXIS ID PB		1.3.2	cmd_hier_axis_id_pb
AXIS ID Text Field			cmd_hier_axis_id
Label Text Field		1.3.2	cmd_hier_label
Hide/Show PB		1.3.2	cmd_hier_hide_show
Special PB		1.3.2	not designed
SC = Single Click			
DC = Double Click			
UP = Move up in list one item			
DN = Move down in list one item			
CUT = Remove item from list			

Figure 10. The XMISE Traceability Matrix: Screen Specification to Design

Lessons Learned

Designing event-driven systems differs from designing procedural systems. It was particularly difficult when time did not permit an exhaustive search of methodologies or training in a new methodology. Part of the system planning phase should have included determining whether the methodologies currently used in house were sufficient for designing the software system. Understanding the type of software system to be developed in the early phases of the system life cycle is crucial. Even though the XMISE designers attended a short course on real-time structured analysis, the course neglected to provide real-world examples of how the methodology could be successfully applied to the design of interactive user interfaces. A single example would have been extremely helpful.

Another lesson learned was the state of the CASE tool industry. While the tools exist, they are not well-integrated and do not provide a full set of functionality. The XMISE team was left with the impression that the available tools (excluding Microsoft Word, MacDraw II, Microsoft EXCEL, and MacProject II) were placed on the market before being field-tested. The CASE tools used for this project constrained the development process instead of enhancing it.

CHAPTER 7. IMPLEMENTATION

As the requirements stated, XMISE was to be implemented using the C programming language, the UIMS, OSF/Motif and the X Window System on an ULTRIX-based workstation. Several issues had to be addressed before coding could begin. Because the first six phases of the project required more time and effort than originally estimated, the implementation phase was severely behind schedule. Fortunately, the original delivery date for XMISE could be postponed for several months because a large contract within the company that was to use XMISE was canceled. Even though the deadline was postponed, the schedule did not permit time to implement the entire system. The XMISE team had to choose the features and functions that could be implemented in the given time frame, but still produce a system that would allow a user to build a user interface.

The Ever-Changing Prototype

As mentioned throughout this paper, a prototype of XMISE had been developed before the XMISE project began. This prototype provided the functionality to create, save, and modify a user interface. It was developed in the C programming language, UIMS, the X Window System, and OSF/Motif. As XMISE progressed through the system life cycle, so did the prototype.

The prototype developers were members of the XMISE development team. Because they intimately knew the pros and cons of the prototype, they were extremely valuable team members and made innumerable contributions to the requirements, screen specification, and design. However, they, too, gained a lot of

information from the outputs of each phase of the Systems Engineering Life cycle. As each phase was performed and review meetings were held, the prototype began to change. Little by little, features that had been agreed on in XMISE meetings would mysteriously appear as part of the working prototype. Clearly the features and functions identified through the Systems Engineering Life Cycle and team effort were so appealing that the prototype developers could not wait to use them. Thus, the prototype was updated to include many of these features. After the XMISE Screen Specification was produced, many of the existing prototype screens began to resemble screens in the specification document.

The ever-changing prototype had advantages and disadvantages. Because it was a working prototype, a few software developers used it to develop user interfaces to their applications. By updating the prototype with the new XMISE features, these features were made available to these software developers. The fact that the prototype continuously changed demonstrates the need and value of following systems engineering principles. By following an orderly process and approaching each phase as a well-integrated team, many excellent ideas, requirements, and suggestions can be incorporated into the system before it is built.

Another advantage of the ever-changing prototype is that the new XMISE concepts could be tested by users to see if they matched their expectations and desires. The concepts could be changed before building the full XMISE system. Because so many features and functions had been coded by the time the implementation phase began, a large subset of the code had been written and provided an opportunity for reuse. This was the greatest advantage as far as the XMISE team was concerned.

Conversely, there were disadvantages to the ever-changing prototype. The original prototype was a baseline design to improve upon. As the prototype changed, it was difficult to differentiate between original prototype features and proposed XMISE features. Also, once an XMISE feature was added to the prototype, the XMISE developers began to question the necessity for implementing XMISE at all. Another disadvantage was that the prototype code changed so rapidly that it could not be reviewed from week to week without reviewing what seemed to be a completely new version of the code. This complicated the task of identifying code that could be reused.

Choosing What to Implement

Because the project was behind schedule and many of the XMISE functions and features had already been implemented in the prototype, the team chose to implement only that portion of XMISE that would significantly improve the prototype.

Two areas of the prototype needed improvement. The first was displaying and manipulating widget hierarchies. The second was specifying and modifying callback resources and general resources for a widget. Both areas required a significant amount of coding. After reviewing the schedule, the XMISE team decided to implement the first improvement and postpone implementation of the second.

Integrating new code into the existing prototype code was challenging. Because the system was a prototype, the code was not documented and still contained bugs. In order to add the improvement, the existing code needed to be thoroughly reviewed to determine where the new code should be added and its impact on the existing code.

This required the prototype code to be frozen so that it could be modified by the XMISE developers and not by the prototype developers.

The decision was made to include the new documented XMISE code in a new file where it could remain separate from the existing prototype code. Not only did the addition of the new code improve the user interface and functionality of the prototype, it helped to resolve existing software bugs.

Updating the Requirements Traceability Matrix

As part of implementing the improvement to the prototype, the XMISE Requirements Traceability Matrix was again updated. This update was performed only for those features that were part of the improvement. For each control flow name in the matrix, the name of the function that actually implemented the control action in code was added to the matrix. The matrix provided a means by which a requirement could be traced from inception, to the screen specification, to the design, and finally to coded functions in which it was implemented.

XMISE Software Test Specification

Before coding the prototype improvement, a draft of the XMISE Software Test Specification was produced. This document included major test groups and tests within each group for the functions of managing and displaying a widget hierarchy. Testing procedures would be developed from this document.

CHAPTER 8. FUTURE XMISE WORK

Several outstanding items will be completed by the end of 1991. The implementation of the improvement to the prototype will be completed and presented at a code walkthrough. Any problems identified at the code walkthrough will be remedied. The XMISE Test Specification will be reviewed and revised. From the final test specification, the XMISE Test Procedures will be generated, documented, and reviewed. Once the test procedures are completed, they will be followed to test the newly implemented code. Once testing is completed, a Test Report will be generated and delivered to the XMISE team. The executable version of XMISE will be released for in-house use along with a user's manual.

Eventually, the rest of the XMISE design must be implemented, especially the portion that allows widget resources to be added and modified. In addition, the remaining portions of the XMISE user interface must be incorporated into the prototype. This may require only minimal effort, since many of the existing screens look similar to the screens presented in the XMISE Screen Specification. A large part of the effort to change the prototype to XMISE involves documenting, walking through, and testing the existing code. This may seem tedious, but it must be done if the system is going to be maintained over a number of years. Part of the effort to update the prototype also involves updating the XMISE documentation. If all of the prototype screens are changed to match the XMISE Screen Specification, this document will not need to be changed. However, the design document should be updated to include the design that is actually implemented, not the design that was supposed to have been implemented. A CASE tool that supports reverse-engineering

may be able to support this effort. Changing the design would also require changing the traceability matrices.

Updating the existing XMISE documentation, as well as creating a programmer's or maintenance guide, will provide system maintainers with all the information they will need to maintain the system.

XMISE is not intended to be a static system. It should evolve as new releases of the X Window System and OSF/Motif software are released. Additionally, it should change to meet changing user needs. If users needs surpass the limitations imposed by the X Window System and OSF/Motif, a new system will need to be built to accommodate these new needs. At that time, the system life cycle for XMISE will come to a close, whereas the system life cycle will begin for the new system.

CHAPTER 9. CONCLUSION

Following the Systems Engineering Life Cycle was rewarding for both the XMISE developers and COMSAT. Even though the XMISE project did not turn out as originally planned, it provided an excellent learning experience for the developers of XMISE and the developers of the XMISE prototype. Applying systems engineering principles to the modification of the XMISE prototype improved the prototype and ensured that the modifications made to it, and especially to its user interface, were designed and developed from a user's perspective. In addition, the documentation produced during each phase of the life cycle will allow the modified XMISE prototype to be better maintained as well as provide guidance for future modifications and enhancements.

COMSAT benefited from the XMISE project not only because it acquired an improved software development tool, but also because the personnel that were involved as supporting XMISE team members were exposed to the Systems Engineering Life Cycle for the first time. Team members experienced the value of working as part of a well-integrated team with members having different areas of expertise. This exposure to the Systems Engineering Life Cycle and experience working with a well-integrated team may be applied to future projects within COMSAT.

The first several phases of the project exceeded the estimated time and effort for two reasons. First, it requires a certain degree of experience to estimate the amount of time required to complete a task. Without this expertise a schedule is meaningless. If the graduate students had previously participated in the entire system life cycle of a

project, the schedule may have been more realistic. The second reason for the schedule overrun is that several of the phases iterated for weeks. Only experience can provide a guide as to when enough iteration is enough. However, the iterations were not without meaning or value. In order for the system to have evolved into its current design, the iterations were necessary.

Another outcome of following the Systems Engineering Life Cycle was finding appropriate CASE tools to support the implementation of each phase in the life cycle. A system engineer's primary product is documentation. This documentation is not standalone; it typically needs to be linked and used between phases in the life cycle. CASE tools should provide this link. As mentioned in this thesis, the state of the available CASE tools was disappointing. Many tools are available, but few support the entire system life cycle or allow interaction between differing tools. In addition, may of the tools are so restrictive that they are not worth using.

One of the greatest outcomes of applying systems engineering principles to the development of a product was learning the value of review meeting and the minutes taken during these meetings. At each review meeting, detailed minutes were taken and subsequently typed and released to team members. The minutes provided were almost as valuable to the project as the documents that were produced. They provided an additional mechanism for tracing decisions made during the life cycle that affected the outcome of the system. They also documented how well an integrated team functions because the minutes captured suggestions, as well as words of wisdom, from more experienced team members.

Following the Systems Engineering Life Cycle also identified areas that needed to be researched, either within COMSAT or the software community as a whole.

Methodologies that are practical to apply yet provide the necessary rigor need to be developed for designing event-driven software systems. Further, testing methods and techniques need to be developed for event-driven systems and interactive user interfaces. These methods and techniques also need to be practical to apply, without requiring excessive funding or time to be allocated to a project.

GLOSSARY

Callback Resources - The events to which a widget will respond together with the name(s) of the source code procedure(s) that will be invoked when an event occurs.

Composite Widget - A widget that may be comprised of other widgets. A composite widget may be comprised of other composite widgets.

General Resources - The physical appearance, placement, and behavior of a widget.

Widget - An object within a graphical user interface that allows the user to interact with the application. Examples of widgets are pushbuttons, scroll bars, pull-down menus, menu bars, labels, text fields, toggle buttons, etc.

Widget Hierarchy - The hierarchical structure of a composite widget.

REFERENCES

ANATOOL. Advanced Logical Software, Inc., Santa Monica, California.

Asbjornsen, Odd. A. 1988. Class handouts from Systems Engineering Principles course. University of Maryland, Spring Semester.

Asbjornsen, Odd A. 1990. Comments made during XMISE review meetings.

Asbjornsen, Odd A. 1990b. Systems Engineering Principles and Practices: Manuscript.

Asbjornsen, Odd A. 1990c. Class handouts from Systems Cost Engineering course. University of Maryland, Spring Semester.

Atallah, George C. 1989. Class handouts from Systems Engineering and Trade-off Analyses course. University of Maryland, Fall Semester.

Baker, James D. 1990. Class handouts from Human Factors course. University of Maryland, Fall Semester.

Blanchard, Benjamin S., and Wolter J. Fabrycky. 1981. Systems Engineering and Analysis. New Jersey: Prentice-Hall, Inc.

Boehm, Barry. 1988. A Spiral Model of Software Development and Enhancement. Computer (May): 61-72.

Eisner, Howard. 1988. Computer-Aided Systems Engineering. New Jersey: Prentice Hall.

Kantowitz, Barry H., and Robert D. Sorkin. 1983. Human Factors: Understanding People-System Relationships. New York: John Wiley & Sons.

MacDraw II Version 1.1. CLARIS Corporation, Mountain View, California.

MacProject II. CLARIS Corporation, Mountain View, California.

Microsoft EXCEL Version 1.5. Microsoft Corporation.

Microsoft Word Document Processing Program Version 4.0. Microsoft Corporation.

Mills, Harlan D., Richard C. Linger, and Alan R. Hevner. 1986. Principles of Information Systems Analysis and Design. New York: Academic Press, Inc.

McCormick, Ernest J. and Mark S. Sanders. 1982. Human Factors in Engineering and Design. New York: McGraw-Hill Book Company.

Odom, J. E. 1990. Using Box Structures for Definition of Requirement Specifications. IBM Systems Journal 21: 59-77.

Open Software Foundation. 1991. OSF/Motif Style Guide: Revision 1.1. New Jersey: Prentice Hall.

Power Tools. ICONIX Software Engineering, Inc. Santa Monica, California.

Rombach, Dieter. 1991. Lectures notes from the Software Development and Design course. University of Maryland, Spring Semester.

Rubinstein, Richard and Harry Hersh. 1984. The Human Factor: Designing Computer Systems for People. United States of America: Digital Press.

Shneiderman, Ben. 1980. Software Psychology: Human Factors in Computer and Information Systems. Boston: Little, Brown and Company.

Shneiderman, Ben. 1987. Designing the User Interface: Strategies for Effective Human-Computer Interaction. United States of America: Addison-Wesley Publishing Company, Inc.

Smith, Sidney L. and Jane N. Mosier. 1986. Guidelines for Designing User Interface Software, Report ESD-TR-86-278. Bedford: MITRE Corporation.

von Mayrhauser, Anneliese. 1990. Software Engineering: Methods and Management. New York: Academic Press, Inc.

Young, Douglas A. 1989. X Window Systems: Programming and Applications with Xt. New Jersey: Prentice Hall.