10/4/26/89 ✓ ǝ (1)

SANDIA REPORT

SAND88—0796 • UC— ~~13~~ 705
Unlimited Release
Printed March 1989

A Review of User Interface Design Techniques With Applications to the Crypto Algorithm Message Processor (CAMP)

Betty P. Chao

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550
for the United States Department of Energy
under Contract DE-AC04-76DP00789

MASTER

SAND--88-0796

DE89 010527

SAND88-0796

A Review of User Interface Design Techniques With Applications to the Crypto Algorithm Message Processor (CAMP)

Betty P. Chao
Digital Subsystems Software Division
Sandia National Laboratories
Albuquerque, NM 87185

ABSTRACT

This report presents a historical perspective of the difficulties associated with user interface design and a review of interface design techniques. Included in the report is an application using rapid-interface-prototyping to the development of CAMP's user interface.

ACKNOWLEDGEMENTS

The author wishes to thank Dr. Douglas J. Riley, Dr. Margaret Hug, Dr. Melissa J. Smartt, and Mr. Mark H. Price for their suggestions and comments on earlier drafts of this report. The author also gratefully acknowledges the support of Ms. Anne M. Chavez in preparation and revision of the report and the tables contained therein.

CONTENTS

TABLES

1. INTRODUCTION

In the recent past, technological innovation was paramount to the success of a computing system. Today, we find that success depends more on well-designed, high-quality user interfaces, because poorly designed interfaces result in a system's disuse or misuse. To facilitate the design of high-quality interfaces, a methodology is required that is capable of presenting user requirements in a concrete and dynamic form, demonstrating concepts visually rather than relying on verbal or written specifications supported by static drawings. Additionally, the methodology must permit iteration of the design process in order to permit simulation, evaluation, and redesign of different candidate interface designs.

The objective of this development report is to detail the implementation of a rapid-interface-prototyping (RIP) methodology into the traditional software development cycle for the Crypto Algorithm Message Processor (CAMP).

1.1 Scope

The focus of this report is on the development of the user interface for CAMP. Development is in terms of its inception, design, evaluation, and subsequent refinements. The impact of the user interface on software design is also included; however, its implementation is discussed in SAND88-0800 (Design and Implementation of the Crypto Algorithm Message Processor Software).

1.2 Organization of Report

The remaining chapters in this report are organized accordingly:

Chapter 2 examines some of the difficulties with past user interface development efforts and the history of techniques aimed at improving the development process.

Chapter 3 presents a rapid-interface-prototyping methodology, detailing features required for an effective prototyping environment. Also included in this chapter are the benefits and limitations of prototyping, along with a list of candidate software projects that would benefit from this methodology.

Chapter 4 reviews the prototyping tools that are currently available. These tools are categorized into large systems (that is, User Interface Management System - like) versus personal computers.

Chapter 5 focuses on the user interface prototyping for CAMP. Included in this chapter are the prototyping process and benefits associated with this project.

Chapter 6 extends the rapid-interface-prototyping methodology to other software project endeavors.

2. INTERFACE DESIGN AND THE SOFTWARE LIFE CYCLE

The user interface design has plagued many software engineers because of the imprecise manner by which user requirements and needs are expressed. The imprecision stems from two opposing ambiguities:

(1) The user knows exactly what he/she wants but is unable to express the requirements precisely.

(2) The user does not know what he/she needs and has no idea of how his/her needs may change later on.

In the weapon command and control community, the users are typically unsure of their needs because of many external influences (e.g., a complex weapon code system and interaction with numerous agencies). These users, at best, have some inklings about their needs. Frequently the users rely on others (e.g., Sandia's system engineers) to determine their needs.

In this chapter, problems associated with user interface development efforts are discussed along with a historical review of software engineering techniques and methodologies that were used with hopes of resolving the problems.

2.1 Difficulties of Interface Development in the Software Cycle

The traditional software life cycle (Boehm, 1981) consists of: (1) requirements analysis and specification; (2) design; (3) implementation; (4) testing; (5) release; (6) maintenance; and (7) retirement.

2.1.1 Requirements Analysis and Specification

The first step in any system's development is the requirements analysis for determining its functionalities. Traditionally, technical problems are researched and analyzed thoroughly. For example, cryptographic algorithms, data flows, message formats, and communication protocols are readily analyzed and thus well formulated and specified precisely. On the other hand, user interface requirements are treated in a cursory fashion. At best, requirements documents specify that the user interface needs to be "user-friendly." The obstacle in this phase of the life cycle is the difficulty in expressing the user interface requirements.

2.1.2 Design

There is an arsenal of techniques, such as data dictionary, data structure, and structure charts, that are used in the design phase. Again, these techniques are aimed at software design and not user interface design.

It becomes apparent that the emphasis of the design process has been concentrated on the technical merits of a software system. The user interface, until recently, is regarded as secondary in importance, resulting in its neglect during the design process.

2.1.3 Implementation

The recent proliferation of publications on user interface design guidelines (e.g., Smith and Mosier, 1984; Schneiderman, 1987) is indicative of a demand for effective and usable interfaces. This demand stems from past practices, whereby programmers implement user interfaces without regard to user's abilities, background, and knowledge (more frequently, lack of knowledge) of the software system under development. Programmers tend to implement interfaces that are tailored for themselves (i.e., programmers not users) and that are over-generalized (i.e., flexibilities and extra features not specified in the requirements are included). Such over-generalizations and unsuitable tailoring, in turn, complicate the interface from the user's perspective. These inattentions to user's abilities are not the fault of the programmers, but rather a lack of savvy in human factors expertise.

Another difficulty in this phase of the life cycle is the lengthy coding time. Even if the interface is virtually impossible to use, software changes will frequently not be incorporated because of scheduling and cost constraints. Instead, training will be burdened with the task of adapting the users to the system. In addition, the user manual must include all the quirks and convoluted paths associated with the user interface.

2.1.4 Testing

In the weapon command and control community, the testing phase is typically the first time that non-development individuals (e.g., military liaison and human factors personnel) have the opportunity to view the user interface of a system. As mentioned above, changes recommended by these and other individuals are met with resistance. As a result, the interface may be far from optimal in terms of understandability, simplicity, and ease of use.

2.1.5 Release

Sandia is typically tasked with training the users of weapon command and control systems. These users return to their respective installations and, in turn, train their operators. This process, coupled with a complicated user interface, has resulted in operators expressing frustration and stress because of difficulties in using the system. These operators are required to perform various operations on military systems that involve weapon security. Additionally, the military has built in a very strong negative incentive of jeopardizing careers for those who perform poorly on weapons operations. Therefore, any unexpected responses in interacting with the software system may have dire consequences.

2.1.6 Maintenance

For a complicated interface, both the training process and the user manual are unable to address all of the possible quirks resident in it. Consequently, as problems arise, ad hoc changes are made to the user manual and the effected pages in the manual are redistributed. This can be troublesome in keeping the manuals consistent and up-to-date.

2.1.7 Retirement

It is evident that complicated user interfaces without regard to user's abilities generate great dissatisfaction among users, particularly among weapon command and control users. It is imperative that interfaces for these users be extremely straightforward, understandable, well defined, and simple; otherwise, the system may be slated for early retirement.

2.2 Software Engineering Techniques for Specifying User Interfaces

Early efforts have relied on story boards using static drawings or displays for characterizing user interfaces. As the field of software engineering progressed, sophisticated specification techniques gained some attention. Currently, User Interface Management System (UIMS) is in vogue and does incorporate many of the rapid prototyping features.

2.2.1 Story Boards

Static renditions of user interfaces are easily mocked up and presented to users. Story boards can be built using 8 and 1/2 by 11 paper; however, software packages are increasingly available for designing static screen facades. Examples of some personal computer based tools are:

<u>IBM Compatible</u>

Application Display
Management System[1]
AutoCAD[2]

---

1. Application Display Management System is a trademark of International Business Machines (IBM) Corporation

2. AutoCAD is a trademark of Autodesk Incorporated

Dr. Halo[3]
PC Storyboard[4]

Macintosh

MacPaint[5]
MacDraw[6]
Super Paint[7]

The obvious deficiency with story boards is the missing dynamic action between the displays and user actions. The ability to dynamically simulate interface designs is by far more authentic and offers users a perspective of realism.

2.2.2 Formal Specification Techniques

These techniques permit precise descriptions of external behavior of a system without specifying its internal implementation. Formal specification techniques are used in many aspects of software development; thus, their extension to user interface specifications is a natural progression.

Most specification languages are based on two formal models: Backus-Naur Form (BNF) (Reisner, 1981) and state transition diagrams (Parnas, 1969). For BNF, an action is associated with a unique grammar rule; that is, whenever that rule applies to the input language stream, the associated action occurs. The deficiency with specification languages is that it is difficult to determine exactly when something will occur after what input tokens have been recognized.

State transition diagramming associates a transition with an action; whenever the state occurs, the system performs the associated action. Sequence is explicit in a state diagram, whereas it is implicit in BNF; hence, there has been extensive research in using state transition diagrams for specifying user interfaces (e.g., Foley and Wallace, 1974; Singer, 1979; Moran, 1981).

---

3. Dr. Halo is a trademark of Media Cybernetics Incorporated

4. PC Storyboard is a trademark of IBM Corporation

5. MacPaint is a trademark of Apple Incorporated

6. MacDraw is a trademark of Apple Inc.

7. Super Paint is a trademark of Silicon Beach Software Incorporated

Both techniques require rigorous and non-trivial notation schemes. The resultant diagrams are difficult to read and understand, thus limiting their usefulness in communicating with users. Additionally, the complexity of the techniques may besiege the designer and detract him/her from the task at hand.

2.2.3 The Wizard of Oz Technique

The Wizard of Oz technique is based upon L. F. Baum's classic novel of the same title. The technique uses an experimenter, hidden from the user, to present and control the user interface prototype. The user unknowingly believes that the ensuing interaction constitutes the real application system. Applications of this technique have been used for development of query-like dialogue interfaces (e.g., Good, Whiteside, Wixon, and Jones, 1984; Green and Wei-Haas, 1985).

The strength of this technique lies in its effective use of iterative design, incorporating feedback from user behavior into the interface. With the hidden experimenter manipulating the interface, user recommendations for improvement can be rapidly incorporated and represented.

There are a number of limitations associated with this technique. First, its use has been limited to problem domains that are small (e.g., electronic mail system and weather reporting), and to users who are unfamiliar with computer applications. Second, it is very difficult to change from one presentation style to another (e.g., from a query dialogue mode to menu selection) in the midst of an interaction between the experimenter and user. Finally, the interplay between the experimenter and user requires certain skills and experience on the part of the experimenter in order to present and modify the interface effectively and efficiently.

2.2.4 User Interface Management System (UIMS)

User Interface Management System (UIMS) is a relatively new area of software engineering research whereby an integrated approach to software design within the traditional life cycle is utilized. UIMS's are interactive systems that support the specification, design, implementation, prototyping, execution, evaluation, and maintenance of human-computer systems, including their interfaces (Hartson, Ehrich, and Johnson, 1986). These systems tend to be quite complex because they include an interface author, run-time libraries, database, and hardware device interfaces that together provide a running system.

The UIMS approach is appealing because of the interface independence feature that separates the user interface from the computational components of the system. This separation has several implications:

1. Several user interfaces may be specified for the same system. This affords the user the opportunity to view different presentations and determine the best one for his/her needs.

2. Code can be easily modified and maintained. Without decoupling the user interface from the computational components, it may be difficult in large systems to locate the code responsible for specific dialogue instances that require modification. Testing these modifications which have to be compiled, linked, and executed is often a time-consuming process.

3. The distinct components can be developed by different specialists for facilitating the development of quality systems (e.g., human factors specialists for developing the user component). There are currently a number of UIMS's being developed in different university environments. Details of some such systems are discussed in Section 4.1 below.

The primary disadvantage of the UIMS, which has limited widespread use, is its complexity. However, with extended interest in applying UIMS to software systems, this approach may be the key to user interface designs in the near future.

2.3 The Interface Design Problem

The user interface design is recognized as a major bottleneck in software system development. Numerous efforts have been devoted by experts in software engineering to effectuate the user interface development. One such effort is the rapid-interface-prototyping methodology (RIP), which is a significant subset of UIMS.

3. RAPID INTERFACE PROTOTYPING (RIP) METHODOLOGY

The basic principle underlying RIP is the iterative process of design, simulation, evaluation, and redesign. User interface design is far from an exact science; therefore, any methodology that does not permit iteration will not capture the evolutionary process, necessary for defining user needs.

3.1 Features of RIP Methodology

Premised on the iterative principle, many proponents of RIP have assembled desirable features required for an effective prototyping environment (Boar, 1984; Johnson, Hartson, Ehrich, Roach, Reilly, Siochi, and Tatem, 1986; Myers and Buxton, 1986; Schwalm, Thomas, White, and Williams, 1987; Rosenberg, Wilson, and Nelson, 1988). The following is a collection of features enumerated by the practitioners:

3.1.1 Non-Programming Interface Author

The interface author is the means by which user interfaces are designed. A non-programming authoring environment allows specialists, who may not be proficient in programming, to readily design the interfaces. By the same token, programmers are alleviated from the task of designing user interfaces which are generally not their primary concerns. Additionally, the non-programming author facilitates the iterative process without the tedium of compiling, linking, and executing code.

The author needs to support the following:

1. <u>The design of displays.</u> The mechanism for generating text, objects, and color should be readily accessible. Also, a superframe (e.g., template) that alleviates regeneration of portions of a display over and over again is desirable. A superframe that can be edited and then overlaid with displays would further expedite the design process.

2. <u>The logic for sequencing displays.</u> The linkage between displays is necessary to provide realistic dynamic simulation of interfaces. The linkage should be sophisticated such that when a menu option or icon is selected from a display, the path of the dialogue will be followed through.

3. <u>The entry of data.</u> There are a number of interfaces that do not require users to enter data (e.g., Sandia's safeguards systems); however, others do require data entry (e.g., Sandia's weapon command and control systems). Data entries need to be stored for feedback later on in the dialogue.

4. The WYSIWYG format. The interface that is being developed should be visible at all times and changes should be immediately apparent.

5. The simulation of dialogue. Once the dialogue along with its paths has been designed, a capability to immediately simulate is required. The simulation must not require any compiling and linking in the traditional sense of programming languages.

6. The change capability. The ease and rapidity in changing displays and paths also facilitate the iterative process. Because time is the critical element here in the iterative process, the designer who can readily switch from the simulate mode to the design mode, make changes easily and quickly, and then simulate the modified dialogue, has a superior communication channel between the users and the system.

3.1.2 Multiple Dialogue Modes

There are many dialogue modes used for presenting interfaces. The selection of the dialogue mode must be based on user requirements. The dialogue design can involve a combination of two or more modes, since different dialogues are appropriate to different tasks and different categories of users (see Chao, 1986 for a discussion of frequently used dialogue modes and the tradeoffs among the different dialogues, user type, and system response).

The interface author which is extensible to the design of different dialogues (e.g., menu selection, form filling, command language, and interactive graphics) will greatly enhance its usefulness. This capability, however, may be quite difficult to incorporate into an effective tool. As discussed in Chapter 4 below, tools are currently available which permit textual or graphical dialogues, not both. Textual dialogues are only suitable for text with some manipulations of, say, an IBM PC's extended character set for generating crude graphics. Graphical dialogues are only suitable for graphics manipulations, because the associated text is also bit-mapped and consequently does not lend itself to ease of use. Perhaps a hybrid of the two types of dialogues can be entertained; for example, a text dialogue author with a library of icons/symbols for presenting limited graphics.

To complicate the tool implementation issue, new techniques, such as windows, are increasingly popular and may quickly become a norm required by users. Most tools are premised on objects; that is, each item whether it is a menu option or an icon is treated as an object. The introduction of a window then requires the generation of another object that is set underneath the menu option or icon. This is a circuitous way to design a window; therefore, in this example, the tool needs to incorporate the concept of either an area or superframe that segments the display, and then design can proceed as usual after the segmentation.

Incorporating up-to-date techniques in an interface author is obviously problematic and not realistic. The designer has to ascertain his/her needs and determine the suitable author for him/her.

3.1.3 Multiple Input Devices

As with multiple dialogue modes, there are many and increasingly more input devices. Although there have been numerous studies on pros and cons of input devices (e.g., Card, English, and Burr, 1978; Karat, McDonald, and Anderson, 1986; Chao, 1987), the appropriate input device for a particular interface is best determined by the user. Again, it is impossible to accommodate up-to-date input devices, but a set of commonly used devices needs to be included. The popularity of direct manipulation devices where the user typically uses a mouse to select and manipulate objects on the screen is becoming predominant for modern computer systems. Therefore, a set would consist of keyboard (including cursor control, keypad, and special function keys), touch, and mouse devices.

3.1.4 On-Line Data Collection

Evaluations of user interfaces require a means for collecting on-line data. Subjective evaluations, such as observing and communicating with the users are not as rigorous as objective evaluations. A data collection scheme, included in the prototyping environment, would facilitate rapid and objective evaluations of user performance. Typical objective performance measurements consist of "time-stamping" and "entries-capturing." Time stamping provides a record of the amount of time that was spent on each display, and thereby gives useful information on display clutter, information overload, etc. Entries-capturing, such as a record of keystroke, touches, and mouse selections and manipulations, would provide information on frequency of menu selection, types of inputs, etc.

In addition to a data collection scheme, either a data reduction and analysis procedure or a utility procedure for formatting the data into a generic data management format (e.g., dBASE[8] format) is desirable. This is because enormous amount of data can be quickly generated, and without the means to reduce the data for analysis, it would be extremely cumbersome for evaluating candidate interface designs.

---

8. dBase is a trademark of Ashton-Tate

3.1.5 Transportability

Transportability refers to the transfer of software from one level or type of system to another. This feature is essential in order to evaluate candidate interfaces on computer systems that are representative of the final product that is to be delivered. Transportability can be as basic as transferring software from one DOS machine to another, or as sophisticated as transferring executable code along with device drivers and emulators for supporting a variety of monitors and entry devices without re-coding the supporting software.

3.1.6 Code Generation

After the interface has been designed, the capability to use it during the implementation process is a desirable feature. An unnecessary step is avoided whenever the prototyped interface can be used as part of the final product. This feature can entail generating the prototyped interface into ASCII text files, or generating actual run-time code.

The weapon command and control projects typically have stringent security requirements; that is, the software requires special design considerations in order to maintain the integrity of classified information and data processing. Therefore, the generation of actual run-time code is not desirable since it may breach security constraints. For this class of software projects, the generation of ASCII text files is sufficient.

3.2 Benefits of RIP

A number of benefits can be realized by the RIP approach. These include:

1. Superior Communication Channel. As mentioned in an earlier section, one of the difficulties in user interface design is the ineffective communication occurring between users and developers. Rapid prototyping permits an interface to be quickly mocked up and thus provides the means by which users and developers can readily discuss the requirements and their ramifications on the outcome of the project.

2. Functional Verification. Although the intent of the RIP approach is to aid the user interface design process, it may also be used for functional verification. Since the user interface reflects the functional capabilities of a system, the prototyped interface can demonstrate that the functions are fully incorporated, and more importantly, that "extra" capabilities are not included.

3. User Satisfaction. Since user participation is essential to prototyping, users take great pride in the final product because of a sense of involvement and accomplishment.

4. Design of Interfaces by Specialists. Specialists, such as human factors individuals, are well versed in the effective design of user interfaces, but are not necessarily proficient in programming. Thus, RIP embodied in an effective tool can be used by the specialists, rather than the programmers, to design the interfaces.

5. Parallel Activities by Developers. Prior to using the RIP approach, weapon command and control projects have progressed in a serial fashion. That is, the systems engineers first perform the requirements analyses, second, the developers design and implement the system, and third, the support specialists (e.g., quality assurance, manual writer, trainer, and human factors) perform their evaluations and duties. This serial process is time-consuming, with insufficient time allocated towards the latter activities. The RIP permits many of the activities to be performed in parallel, particularly the support activities. Also, an added benefit is that the recommendations stemming from the support evaluations can be readily incorporated into the design.

In addition to the benefits gained within the software engineering realm, a "business case" can be realized by the RIP approach. Future systems can be easily marketed to determine their feasibilities, and since revisions are quickly completed, the prototyped interface can be used for attracting a wide range of customers.

3.3 Limitations of RIP

The RIP methodology is not a panacea for all user interface designs. In fact, it is most suitable for data processing systems with limited potential for real-time processing systems. Also, with the ever changing technology associated with entry devices, any RIP tool would become obsolete quickly.

There are limitations associated with rapid prototyping itself. Large projects may result in unwieldy interfaces with the evaluations difficult to manage and control. The prototyped interface may give the illusion that the project has progressed considerably and that the final product is imminent and shortly forthcoming; whereas, in actuality the system has yet to be fully analyzed and perhaps not even designed. For the business case, marketing may oversell the prototyped interface, creating unrealistic expectations for the actual product performance.

The significance of rapid prototyping is simply a methodology aimed at improving the user interface design process. The effectiveness depends on type, size, and application of the software system. The designer needs to determine whether rapid prototyping is appropriate for his/her needs by weighing the pros and cons of the methodology.

4. TOOLS AVAILABLE FOR IMPLEMENTING RIP

There are a number of tools available in both the commercial market and university environments. These tools can be classified into UIMS-like systems/workstations and personal computers (PCs). Most of the UIMS-like systems are found in university environments with limited commercial dissemination, whereas the PC tools are generally commercially available.

This chapter highlights some of the tools, and is by no means a comprehensive review of all the tools currently available. Because there is an increasing number of new prototyping tools entering the market, a thorough review is not feasible. The emphasis here is to provide the reader with a flavor of the more well-known tools.

4.1 UIMS-Like Tools

These tools tend to be general purpose tools on workstations (e.g., graphics workstations, symbolics workstations, and distributed processing with micro-computers). Table 1 lists some of the more common UIMS-like tools along with a comparison of the features that the tools support (e.g., non-programming environment, types of input devices, types of dialogue modes, hardware).

4.2 Microprocessor Based Tools

There is an increasing number of prototyping tools entering the market for personal computers. They range from simple story boards for drawing on the CRT screens to interactive prototyping products. Some of the commercially available story boards are listed in Section 2.2.1. Table 2 lists five of the more frequently used tools for IBM and Apple's MacIntosh personal computers.

Table 1: UIMS - Like Tools on Workstations Used for Rapid Prototyping

	Non-Programming Environment	Supports Dialogue Modes		Supports Input Devices					On-Line Data Collection	Suggested Hardware	Code Generator	Comments
		Text	Graphic	Key	Touch	Mouse	Digitizing Tablet	Voice				
Rapid/Use		X		X		X			X	SUN/Unix VAX/Ultrix VAX/VMS Apollo/Unix	X	* Uses Transition Diagrams to design and sequence interface. * Provides modular user interface.
COUSIN			X	X	pointing device					VAX/Unix Perq/SPICE		* Uses editor to generate display. * Provides modular user interface.
Peridot	X		X	X						Xerox 1109 AI Workstation	X	* Specifically intended for use with direct manipulation devices (simulated mouse). * Generates Interlisp-Dcode. * Provides modular user interface.
Trillium	X		X	X		X				Xerox 1186 AI Workstation		* Specifically intended for design and simulation of Copiers.
BLOX			X	X		X	X			Bit mapped Workstation ASCII terminals		* Uses State Transition Syntax and Graphics Editor to design and sequence interface.
VAPS			X		X			X (only for flight simulation)		IRIS Workstation		* Uses a mix of menu driven and object oriented programming to design and simulate interface.
AIDE	X	X	X	X	X				X	VAX 11/780	X	* Provides modular user interface. * Requires user to enter conditional expressions in order to simulate interface.

Table 2: PC - Based Tools Used for Rapid Prototyping

	Non-Programming Environment	Supports Dialogue Modes		Supports Input Devices					On-Line Data Collection	Suggested Hardware	Code Generator	Comments
		Text	Graphics	Key	Touch	Mouse	Digitizing Tablet	Voice				
DEMO II	X	X		X						IBMPC/ MSDOS		* Difficult to learn and the manual is very terse; must purchase a separate tutorial in order to learn how to use it. * Limited support of data entries.
Skylights	X				X	X	X		X	IBMPC/ MSDOS	X	* Uses a direct manipulation editor similar to commercial "draw" programs to design screens. * Difficult to learn and the manual is very terse. * Keystroke files are not formatted (time is in hex and keystrokes are coded).
Mirage	X	X		X	X	X			X	IBMPC/ MSDOS		* Designed specifically for support of Sandia's command and control systems.
HyperCard	X	X		X		X				MacIntosh	X	* Uses a direct manipulation editor to design screens. * Requires programming skill to link displays.
Prototyper	X	X		X		X				MacIntosh	X	

5. AN APPLICATION: RAPID PROTOTYPING THE USER INTERFACE FOR CAMP

The user interface design process for CAMP was accomplished with Mirage, a prototyping tool developed specifically for Sandia's weapon command and control systems. This chapter details the CAMP user interface design process; whereas, the implementation process is enumerated in SAND88-0800.

5.1 Requirements

5.1.1 System Requirements

CAMP is used for generating files of Code Activated Processor (CAP-MC3764) ciphertext containing Set Weapon Identification and Configure data, and for transferring data to and from the T1563 Automated PAL Controller. The functions associated with CAMP are:

1. Generation of Set Weapon-ID Messages. This function permits a file of Set Weapon-ID Messages to be generated such that the contents of this file can be used by the T1563 for setting Weapon Identifications (i.e., Weapon Mark, Modification, and Serial Number) into CAP-equipped weapons.

2. Generation of Configure Messages. This function permits a file of Configure Messages to be generated for W82-0 weapons only. These messages contain CAP-encrypted data and are used by the T1563 for setting W82-0 weapons.

3. Add a Serial Number or PSA Identifier. This function permits the user of CAMP to interactively add either a serial number or a PSA identifier to the source files.

4. Transfer Data To/From the T1563. This function permits the above files to be transferred to the T1563 and permits the National Security Agency's configure seed files to be transferred to CAMP using the T1563.

CAMP requirements are detailed in CD384275 (Compatibility, Crypto Algorithm Message Processor).

5.1.2 User Requirements

CAMP is intended for use by operators at Mason & Hanger in Amarillo, Texas. These operators are those involved in the weapon production lines. They are not likely to be familiar with computing systems or computing terminologies. Thus, the user interface must be (1) well defined, (2) extremely straightforward, (3) understandable, and (4) simple. Furthermore, the training requirements must be minimal so as to reduce the potential for errors.

5.2 Design and Evaluation of the CAMP Interface

An initial user interface prototype was developed on an IBM-PC using Mirage (see McDonald, Vandenberg, and Smartt, 1987 for a detailed description of the Mirage prototyping tool). The prototyped interface was a general conceptualization of the user's requirements for preparing files of CAP messages. It was menu driven and contained approximately 25 screens. Upon viewing the interface, the user and Sandia engineers were able to readily communicate in detail the necessary steps for preparing these files; that is, details of the source files, output files, and an editing feature for modifying the source files. The objectives during this phase were to resolve ambiguous requirements and to provide a model of the interface that behaved as the user expected.

The second interface was again menu driven with numerous interactive keyboard entries for specifying various filenames and memory phrases to decrypt and encrypt contents of the files, and for using the editor. This interface contained over 70 screens. For this phase of the evaluation, the user enlisted the aid of their systems engineers who were accustomed to designing user interfaces for their operators. At this phase, the basic requirements were defined, but the "how-to" implementation of the requirements needed to be developed. Here, the user's systems engineer provided invaluable recommendations with respect to the following:

1. Display Layout. The user identified the optimal locations on the screen for presenting menu items, instructions, and miscellaneous information such as audit trails.

2. Keyboard Entry. The user requested minimal keyboard entries which greatly influenced the design of the software. Since the user did not want their operators to type in filenames, default filenames for both source and output files had to be used. Also, the source files had to reside in different diskettes, one for the Set Weapon ID source file and another for the Configure Messages source file, in order to accommodate the use of default filenames which needed to be transparent to the operators.

3. Sequencing of Displays. The user provided recommendations as to the sequence of displays with aborts interspersed throughout the dialogue. Additionally, the user simplified the editor to merely an "add" feature. This was because the flexibility of a full editor was deemed too complex to operate so that the editor was stripped down to a minimum.

4. Wordings and Terminologies. Instructions, descriptions of menu choices, and audit trails were rephrased to reflect the vocabulary used by the operators. Verbosity was reduced such that the dialogue can be read quickly and understood at once. Unfamiliar terminologies were replaced with familiar terminologies (e.g., transfer of data from CAMP to T1563 instead of PDM emulation).

It is interesting to note that by the end of this phase, both the user and Sandia engineers were realizing the CAMP requirements and their impact on user's operations. Even though the intent of the prototyping process was aimed towards defining the user interface, it provided an excellent communication vehicle for all the parties. Once the user saw how the originally requested features (e.g., a full editor) increased both operator work-load and the complexity of the interface, these "extra" features were quickly eliminated.

The third interface was menu driven with minimal keyboard entries. This interface contained approximately 30 screens. For the third iteration, the user enlisted their operators, the ones who will be performing the operations on CAMP, to evaluate the interface. The operators suggested refinements for specifying instructions on the screen; that is, they wanted the instructions to be patterned after their own "Operations and Instructions" manuals.

The outcome of this iterative design and evaluation process is a tailored user interface that provides only the necessary operations for preparing files of CAP messages and subsequent transfer of these files to another piece of equipment, the T1563.

5.3 Benefits of Rapid Prototyping CAMP's Interface

The RIP approach was beneficial to all parties. The users expressed great satisfaction with the software product. This was because of their intimate involvement with the user interface design. Additionally, the product delivered was as expected such that the users were not caught by any surprises or besieged by an unfamiliar product.

For the designers, the benefits were numerous:

1. In the past, the requirements definition occurred at the same time as design which complicated the design process because of changing requirements. The RIP process facilitated the requirements definition such that functional requirements were all known before the start of the development process.

2. Since all parties subscribed to the user interface, it remained stable throughout the entire project. The stability permitted ease in designing and implementing the software.

3. The user interface provided the framework for the software design since it embodied all the functional requirements.

4. The developers addressed error conditions and recovery procedures early in the development process. This was possible because the user interface permitted one to visualize different

types of errors as one dynamically stepped through each of the displays.

5. The test plan was developed in parallel with the development activities. With the user interface and the requirements document at hand, test matrices and test cases were readily developed. For each of the test cases, the expected outcome was directly traceable to a particular display of the user interface.

An additional benefit that was not realized with the CAMP project, but would be very prominent in future weapon command and control projects, is the assimilation of Sandia's support activities. The support activities, such as human factors evaluation, training, manual writing, and quality assurance, are typically performed after the product prototype has been developed (close to the end of the development process). With the user interface at hand, the support activities can occur in parallel with the development process, allowing more of their recommendations to be incorporated into the product and most importantly, allowing a timely release of the product.

6. CONCLUSIONS

The RIP methodology is now suggested as a guideline for Sandia's software developments. Mirage is currently being used to develop interface prototypes for several command and control software products under development. Other sectors of Sandia use other tools, such as Hypercard,[9] for developing interface prototypes. Regardless of the tool used to implement the RIP methodology, the success of the methodology has been demonstrated and is widely accepted, and consequently recommended as part of Sandia's software development process (e.g., Schroeder and Cooper, 1988).

9. Hypercard is a trademark of Apple Corporation

7. REFERENCES

Boar, B.H. (1984). Application Prototyping. New York, NY: John Wiley and Sons.

Boehm, B.W. (1981). Software Engineering Economics. New Jersey: Prentice-Hall, Inc.

Card, S.K., English, W.K., and Burr, B.J. (1978). Evaluation of Mouse, Rate-controlled Isometric Joystick, Step Keys, and Task Keys for Text Selection on CRT. Ergonomics, vol. 21, pages 601-613.

Chao, B.P. (1986). Design Guidelines for Human-Computer Dialogues. SAND 86-0259, Sandia National Laboratories, NM.

Chao, B.P. (1987). Prototyping a Dialogue Interface: A Case Study. In G. Salvendy (Ed.), Cognitive Engineering in the Design of Human-Computer Interaction and Expert Systems. New York, NY: Elsevier Science, pages 357-364.

Foley, J.D. and Wallace, V.L. (1974). The Art of Graphic Man-Machine Conversation. Proceedings of the IEEE, vol. 62, no. 4, pages 462-471.

Good, M.D., Whiteside, J.A., Wixon, D.R., and Jones, S.J. (1984). Building a User-Derived Interface. Communications of the ACM, vol. 27, no. 10, pages 1032-1043.

Green, P. and Wei-Hass, L. (1985). The Rapid Development of User Interfaces: Experiences with the Wizard of Oz Method. Proceedings of the Human Factors Society - 29th Annual Meeting. Santa Monica, CA: The Human Factors Society, pages 470-474.

Hartson, H.R., Ehrich, R.W., and Johnson, D.H. (1986). Introducing Dialogue Management. In Ehrich and R. Williges (Eds.), Human-Computer Dialogue Design. New York, NY: Elsevier Science, pages 11-107.

Johnson, D.H., Hartson, H.R., Ehrich, R.W., Roach, J.W., Reilly, S.S., Siochi, A.C. and Tatem, J.E. (1986). The Dialogue Author. In R. Ehrich and R. Williges (Eds.), Human-Computer Dialogue Design. New York, NY: Elsevier Science, pages 109-163.

Karat, J., McDonald, J.E., and Anderson, M.A. (1986). A Comparison of Menu Selection Techniques: Touch Panel, Mouse, and Keyboard. International Journal of Man Machine Studies, vol. 25, no. 1, pages 73-88.

McDonald, J.E., Vandenberg, P.J., and Smartt, M.J. (1987). The

Mirage Rapid Interface Prototyping System. MCCS-87-87, New Mexico State University, Las Cruces, NM.

Moran, T.P. (1981). The Command Language Grammar: A Representation for the User Interface of Interactive Computer Systems. International Journal of Man-Machine Studies, vol. 15, pages 3-50.

Myers, B.A. and Buxton, W. (1986). Creating Highly-Interactive and Graphical User Interface by Demonstration. Proceedings of ACM SIGGRAPH '86, pages 249-258.

Neidigk, D.N. (1988). Design and Implementation of the Crypto Algorithm Message Processor Software. SAND88-0800, Sandia National Laboratories, NM.

Parnas, D.L. (1969). On the Use of Transition Diagrams in the Design of a User Interface for an Interactive Computer System. Proceedings of the 24th National ACM Conference, pages 379-385.

Reisner, P. (1981). Formal Grammar and Human Factors Design of an Interactive Graphics System. IEEE Transaction on Software Engineering, vol. 7, pages 229-240.

Rosenburg, D.J., Wilson, J., and Nelson, M.A. (1988). Rapid Prototyping for User Interface Design. Workshop for ACM/SIGCHI '88.

Schneiderman, B. (1987). Designing the User Interface: Strategies for Effective Human-Computer Interaction. Reading MA: Addison-Wesley.

Schroeder, D.H. and Cooper J.A. (1988). Weapon Microprocessor System Design Standards Committee Final Report. SAND88-0423, Sandia National Laboratories, NM.

Schwalm, R.C., Thomas, M.E., White, R.J., and Williams, R.D. (1987). User Interface Prototyping in Military Systems. Proceedings of the National Aerospace Electronics Conference, pages 247-254.

Singer, A. (1979). Formal Methods and Human Factors in the Design of Interactive Languages. Ph.D. dissertation, Computer and Information Science Department, University of Massachusetts.

Smartt, M.J. (1987). Compatibility, Crypto Algorithm Message Processor (CAMP). CD3842705, Sandia National Laboratories, NM.

Smith, S.L. and Mosier, J.N. (1984). Design Guidelines for the User Interface for Computer-Based Information Systems. The MITRE Corporation, Bedford, MA.

Distribution:

2300	J. L. Wirth
2310	M. K. Parsons
2312	D. J. Allen
2312	A. E. Farmer
2312	J. D. Mangum
2315	M. J. Smartt
2315	D. R. Blazek
2315	B. P. Chao (50)
2315	A. M. Chavez
2315	S. N. Giles
2315	M. A. Hug
2315	D. D. Neidigk
2315	M. H. Price
5126	W. D. Chadwick
5126	J. W. Bonahoom
5126	J. F. McDowell
7223	R. G. Easterling
7253	R. E. Baack
7253	J. Huttenhow
3141	S. A. Landenberger (5)
3141-1	C. O. Ward (8) (For DOE/OSTI)
3151	W. I. Klein (3)
8524	J. A. Wackerly