An Evaluation Of Three User-System Interface Specification Techniques

by

William W. Smith III

Thesis submitted to the Faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Industrial Engineering and Operations Research

APPROVED:

Robert C. Williges, Chairman

W. J. Fabrycky

Wolter J. Fabrycky

Beverly H. Williges

February, 1988

Blacksburg, Virginia

AN EVALUATION OF THREE USER-SYSTEM INTERFACE SPECIFICATION TOOLS

by

William W. Smith III

Committee Chairman: Robert C. Williges
Industrial Engineering and Operations Research

(ABSTRACT)

User-System Interface (USI) design is a highly iterative process involving empirical testing and evaluation. The existence of a design specification is implicit in this design process. The quality of the design specification impacts the length and cost of the design cycle. A survey was made of available USI specification tools to determine: which USI components they can specify; their relation to a finite state model of the USI; and if they are usable by the human factors engineer as a basis for an analytic analysis of a USI design to detect violations of excepted USI design guidelines. Four categories of tools were surveyed: semi-formal techniques, Backus-Naur Form grammars, programming languages, and transition networks. An engineering tradeoff analysis was performed based on four pragmatic criteria: understandability, efficiency, expressive power, and fidelity.

The results of the tradeoff analysis suggest that specification tools differ in representational strength and it is best to use a set of tools for a complete specification. A behavioral study using human factors engineers was performed to validate the results of the tradeoff analysis. Human factors graduate students were trained in the use of one of three specification tools and asked to perform an analytic analysis looking for design defects. Fifteen design defects were imbedded in each of the specifications. Results suggest that using two of the tools together is the most effective specification technique. In light of these results, current and future software design practices were examined to determine what role the human factors engineer can have on the design team.

Acknowledgements

Support for this thesis was provided under subcontract to the Federal Systems Group of TRW, Inc., on Proposal 8617626 "RAMCAD Software Development" I wish to thank my committee members Dr. Robert C. Williges, Dr. Wolter Fabrycky, and Bev Williges for their intellectual support and guidance. Their understanding of my difficult circumstances ensured the completion of the thesis.

There were many others involved in the effort. Without their support the thesis would not have been possible. To mention just a few: Scott P. Overmeyer and Kenneth B. Gaylin, Federal Systems Group, TRW Inc., for assistance with the rapid-prototype specification; Cary Moon for editorial advice and proof reading; Octavio Torres for intellectual energy and for assisting in the design of the experiment; and the graduates students in the Human-Computer Interaction Lab for pushing me to finish. Finally, I wish to thank Lucy Wakefield Smith and William W. Smith for their patience and lasting strength.

Table of Contents

List of Tables

List of Illustrations

CHAPTER 1. INTRODUCTION

The major trends in computer technology over the past few decades are apparent: increasing speed, increasing reliability, decreasing size of components, and decreasing unit cost (Nickerson, 1986, p.9). The impact of these trends on the type of computer user have been great. The emergence of the personal computer has made computing power available to writers, store clerks, and technical non-computer specialists such as engineers. It is clear that just about everyone will make use of a computer in some way.

Accompanying these rapid changes in the power of the computer and the type of user, there have been major shifts in the way computer systems are designed. Early in the history of computers, users were specialists with intimate knowledge of the internal operations of computers. Knowledge of this kind was required to perform useful work with these systems. This group of users has very different information needs than that of an electrical engineer designing an electronic device using a computer-aided design system (CAD). Driven by technology and market forces, the focus of design has shifted from making a *useful* computer system to one that is also *usable* by non-specialists. Shackel (1984, p.53) distinguishes these two terms: "*useful* means advantageous, profitable, fit for some desirable end, or having (latent) power to satisfy human wants; *usable* means able to be used, applicable to a purpose (i.e., the latent power to satisfy human wants can be harnessed)".

The discipline of Human Factors is concerned with the usability of machines and systems. Those researchers involved with the usability of computer systems study what is frequently called human-computer interaction. Many of these researchers have proposed design methods that lead to improved computer systems that are easier to use. Typical of these recommendations, are those of Gould and Lewis (1985, p.300). They recommend three principles for the design process:

1. ***Early Focus on Users and Tasks.*** First, designers must understand who the users will be. This understanding is arrived at in part by directly studying their cognitive, behavioral, anthropometric, and attitudinal characteristics, and in part by studying the nature of the work expected to be accomplished.

2. ***Empirical Measurement.*** Second, early in the development process, intended users should actually use simulations and prototypes to carry out real work, and their performance and reactions should be observed, recorded, and analyzed.

3. ***Iterative Design.*** Third, when problems are found in user testing, as they will be, they must be fixed. This means design must be iterative; their must be a cycle of design, test and measure, and redesign, repeated as often as necessary.

This process of design can be very costly if many iterations are needed to be performed before a design is completed. A computer system needs to be well defined before simulations and prototypes can be constructed and used for empirical measurement. The existence of an initial design specification is implicit in the design process described above. The quality of this specification will impact the length and cost of the design cycle. A robust and rigorous specification will reduce the number of iterations through the design process.

Carroll and Rosson (1985) believe that a design specification should include both functional (the functions to be performed by the user and/or the computer system) and usability specifications. These specifications should be motivated by both functional and usability objectives, with the former motivating the functions substance, and the later motivating the functions form. Carroll and Rosson give an example: "... the functional objective of 'document creation' could combine with the usability objective of 'little time for training', to produce a specification for a menu-driven interface for the document creation process" (p.15). The logic behind selecting a menu-driven interface is that the user is not asked to memorize lengthy commands that require in depth training. Like Gould and Lewis (1985), Carroll and Rosson see the next stage in the design process, after initial specification, as usability testing. What is not clear in both studies is how a good initial specification is developed and what form it should have. Carroll and Rosson say only that the *details* of the specification will be largely a function of intuitive, empirical, and analytic work of the designer.

The *details* of a system design will have a large impact on the usability of the system during empirical testing and iterative development. There is a very large set of attributes of a computer system that the designer can vary (e.g., input devices, color and size of text in the display, modes and states of the system software). Most of these attributes have to be specified and design decisions made about them before empirical testing can proceed. If the specification is not analyzed and specified in as complete a way as possible before empirical testing, it may be very problematic to determine from the empirical test what part of the design is responsible for usability difficulties. An initial design specification should have as many as possible potential usability problems corrected analytically before expensive and lengthy empirical testing is performed. A design specification should be critically examined before empirical testing to see if it violates basic human factors principles (e.g., *consistency* [Williges, Williges, and Elkerton, 1987, p.1420]) and agrees with recognized design guidelines (e.g., Smith and Mosier, 1986).

From a practical viewpoint, it is unlikely that problems of usability uncovered in empirical testing will be solved by radical departures from the initial design specification.even if they are necessary. It is more likely that there will only be iterative refinement on the original design. In fact, it is known from the human decision making literature that once an initial hypothesis is formulated, evidence is sought mainly to confirm it not to refute it (Wikens, 1984). This is known as a confirmation bias. Also, humans fail to use negative evidence in decision making. These findings have implications for the effectiveness of empirical measurements. Even when the "writing is on the wall" and a computer system should be reworked completely because of evidence in usability testing, the designers will not reject the initial design, but just refine it in minor ways.

PURPOSE

The conclusion that can be drawn from the this discussion is that it is important to perform an analytic human factors analysis of the design specification before empirical testing and possibly throughout the design process too, since errors may actually be

introduced into the design during iterative development and refinement. The purpose of this thesis is to determine what system specification tools are available to the designer and to recommend one(s) that is(are) best suited to the task of human factors analysis.

The thesis is divided into four additional major sections. The first section describes the components of the user-system interface (the interface being the place where the user and the computer system meet). The characteristics of these components must be specified by the designer in the initial design stages. In the second section of the the thesis, a survey and an engineering tradeoff analysis of system specification tools is conducted to determine which of these tools can describe the components of the interface and have desirable characteristics for human factors analysis. The third section describes the procedure of a behavioral study conducted to validate the results of the tradeoff analysis. In the study, human factors specialists were asked to perform an analytic analysis of a computer interface specified with a tool selected from the tradeoff analysis. Overall conclusions and recommendations are provided in the last section.

CHAPTER 2. COMPONENTS OF THE USER-SYSTEM INTERFACE

Before interface specification tools can be compared and described, it will be necessary to define the components of the interface that they attempt to specify. In the discussion that follows, the terms *user-system interface* and *interface* will be used interchangeably. The user-system interface (USI) is the more precise term. *User* in this case is the person for whom the computer system is being developed to meet specific needs. The *system* is the computer and its components. These components include the input devices (e.g., keyboards, light pens, speech recognizers), the output devices (e.g., video displays, audio speakers, printers), and finally the programming logic that controls how the system behaves. The *interface* is then the place where the user and the system have a dialog.

Attempts by human factors researchers to define the components of the USI in terms of taxonomies have drawn heavily on language based models of human communication developed in the field of linguistics. A taxonomy being a means of classifying objects or phenomena in such a way that useful relationships among them are established. Linguistics is the field concerned primarily with the study of human language and communication. Many of the properties of person to person communication are shared by person to computer dialogs. Taxonomies are useful to the design, specification, evaluation and comparison of USIs. They allow breaking of the interface into components facilitating critical examination and avoid comparing interfaces at incompatible levels of detail (the "apples and oranges" problem). Additionally, a taxonomy can be used to organize and classify interface specification tools so one can see what components of the USI they are capable of specifying. First, two existing taxonomies are presented, and their ability to represent all the components of the USI is examined. Then by the

synthesis of the existing taxonomies and relevant issues, a new taxonomy is generated that will be used to classify interface specification tools.

TWO USER-SYSTEM INTERFACE TAXONOMIES

Moran's Taxonomy

Moran (1981) proposes a three component (Conceptual, Communication, and Physical) taxonomy of the USI. Each component contains levels which are more detailed. The taxonomy is hierarchical, starting with user's cognition and moving toward properties of the interface and the workstation. The taxonomy emphasizes the cognitive properties of the user. The three taxonomy components and their levels are described below.

Conceptual Component. This component contains the abstract concepts around which the system is organized. The notion is that the user forms a *conceptual model* of the system from these abstract concepts. This model is more or less a definite representation or metaphor which guides the user's actions and helps her to interpret the system behavior. The conceptual component has two levels, the task level and the semantic level. The task level represents the purpose of a system. It is a structured representation of the tasks the user will set for herself with the aid of the system. At this level, some assumptions must be made about the functional power of the system, but not its specific features. The semantic level contains representations of the conceptual entities of the system and the conceptual operations on these entities. (*Semantics* means relating to the meaning, reference and truth of statements.) This level defines the connections between the system's functional capability and the task domain. System functions are defined only in an abstract way.

Communication Component. This component defines both the command language and the conversational dialog between person and computer. While the conceptual component does not define exactly how system operations are evoked, the communication component defines the commands and user actions needed to tell the computer what operation to perform. There are two levels, the syntactic level, and the interaction

level. The syntactic level contains descriptions of the commands, arguments, contexts and state variables of the command language. Linguistics defines syntax as the rules by which correct sentences are formed. Note that a sentence (or a computer command) can be syntactically correct but not be meaningful. It is in the semantic level that the meaning of a command is defined. The interaction level relates the physical actions of the user and the display actions of the system to the rules governing the dialog. The syntactic level is not enough to specify the command language of the system completely. It is necessary to describe the order in which commands are specified by the user and the specific user actions needed to specify commands and arguments.

Physical Component. The characteristics of the physical devices that the user sees and comes in contact with are defined in this component. The physical component is the least well defined and developed of the components in Moran's taxonomy. There are two levels, the spatial level and the device level. The spatial level defines the arrangement of input and output devices and the arrangement of the display graphics on the video display. The device level defines any remaining physical features of the system.

If Moran has correctly captured all the components and levels of the user-system interface, then, any design specification should have defined aspects of the interface across all parts of Moran's taxonomy. However, as is shown in the next section, Moran's taxonomy has several weaknesses.

Foley and VanDam's Taxonomy

The second taxonomy, is one described by Foley and VanDam (1982, pp.220-222). They view the taxonomy as defining the components of the user-system interface that need to be designed. The taxonomy is seen as a top-down definition of an interface design process.

There are four hierarchical parts to this taxonomy: conceptual, semantic, syntactic, and lexical. Although these terms are used differently, the first three parts are defined similarly to the components and levels in Moran's (1981) taxonomy. The major difference, and the one which deserves attention here, is Foley and VanDam's inclusion of a lexical level.

Lexical Level. This level describes how input and output tokens (the words in a language) are formed from the available system hardware. From the designer's viewpoint, the lexical level translates into a design objective. The designer must design the techniques that the user uses to provide input to the system, and design the primitive shapes (such as lines and characters) and their attributes (such as color and font) that the system displays as output. It is useful to think of user-system communication as consisting of two languages. One is the language used by the user to communicate with the computer, expressed as actions applied to various input devices (such as a mouse or a keyboard), and the other is the language the computer uses to communicate with the user.

The two taxonomies use similar levels of detail to break up the interface, and both rely heavily on terminology used in language models of communication. Which taxonomy is best suited to represent the components of the USI? Buxton (1983) critically examines these two taxonomies and identifies short-comings in both of them. The criticism focused mainly on their under-representation of pragmatic issues.

Pragmatics, as defined in linguistics, is the study of language use and its relation to language structure and social context (Akmajian, et al., 1984). In human communication, pragmatic issues are quite often those anomalies not amenable to formal analysis. Issues such as why the response "Yes" is not an appropriate response to the question "Can you pass the salt?" and why the response "I'm not wearing a watch" can be appropriate as an answer to the question "What is the time?" are examples of pragmatics in linguistics (Norman and Draper, 1986, p.315 f.). One can see that pragmatic phenomena, although they can have a large impact on communication, are concerned with *low-level* things such as the nuances of a small word like "Yes". Similarly, discussions about the pragmatics of user-computer dialogs focus on *low-level* physical characteristics such as:

- where items are placed spatially on the display,
- where devices are placed in the workstation,
- and the type of physical gesture used to articulate a token ("word").

However, the impact of pragmatics on larger issues such as the user's conceptual model of the system, and the form of an interface design, can be great. The importance of carefully designing and specifying the physical aspects of a system is receiving increased attention. Specifically, Buxton(1983) takes a very strong position and writes that physical aspects may have, "one of the strongest effects on the user's perception of the system" (p. 32). As an example, imagine trying to use the Macintosh computer without a mouse. So in any taxonomy for defining the components of the USI, it is important to include pragmatics explicitly. Moran's physical component and Foley and VanDam's lexical level can be looked on as attempts address pragmatic issues. Working from the two taxonomies and the criticisms of Buxton, a revised taxonomy is defined. The language of a taxonomy should not restrict it just to issues of design, but also be useful for issues of description, analysis, specification, implementation and comparison of USIs.

A SYNTHESIS OF USI TAXONOMIES

In keeping with Moran (1981), the taxonomy retains the communication and physical components, but modifies the levels in the communication and physical components. Below is a description of the taxonomy. See the previous section for more detailed definitions of components and levels that are unchanged.

CONCEPTUAL component

Task level. the tasks the user will set for herself with the aid of the system.

Semantic level. the conceptual entities of the system and the conceptual operations on these entities.

COMMUNICATION component

Syntactic level. the commands, arguments, contexts and state variables of the command language.

Lexical level. the ordering of lexemes (the smallest units in a language) and the nature of the alphabet used (symbolic or iconic for example).

PHYSICAL component

Spatial Layout level. the spatial placement of items on the display and the details of their attributes. The placement of devices in the workstation.

Auditory Layout level. the dimensions of sound such as pitch, volume, and timbre. In the case of voice output, the rate of speech and apparent gender of the voice may be issues (Simpson et al., 1985).

Device level. the control characteristics of the input device employed by the user to articulate a token.

When a comparison is made between this taxonomy and that of Moran (1981), two significant differences are apparent in the communication component. First, a lexical level (similar to Foley and VanDam's, 1982) has been added; and the Interaction level has been eliminated. The addition of the lexical level is in keeping with the conventions of the linguistic view of the USI. Moran, in pointing out limitations of his taxonomy, suggests that the addition of a lexical level between the syntactic and interaction levels may be required to reduce the scope of the interaction level, which subsumes many lexical issues and is broader than other levels in the taxonomy (p. 35). And as Buxton (1983) points out, there is not a slot in the Moran taxonomy that one can use to address lexical issues as they are defined above.

More controversial perhaps, is the elimination of the interaction level. The interaction level is actually another viewpoint of the interface rather than the language model used loosely here. Its inclusion mixes the model of interface behavior with it components. The interaction level can be thought of as a "black box" description of the system. At this level, the system is described by the set of possible inputs, dialog control rules (the "black box" part) and the set of outputs. In a sense, it is a holistic view representing a model of system behavior that is a combination of several levels of the USI component taxonomy. This study requires the separation of the two views, while Moran needed to include them together to serve other purposes. In further discussions about interface specification tools and human factors analysis, the behavior of the interface is separated from its components.

The physical component has been expanded to include an auditory level. There is increasing interest in expanding the bandwidth of user-system communication by using sound as a significant part of input and output. There is a need to extend this research and heighten awareness of the applications of sound. There is a definite lack in both taxonomies of a sound component along with a definition of its role in human-computer

communication. The device level was more precisely defined as dealing with the physical characteristics of input devices to emphasize their impact on user actions and perceptions of the system. The physical characteristics of the input devices effect the pragmatics of the user-computer dialogue to such a degree that they should be precisely specified so they may be analyzed.

Specifying components of the USI taxonomy is not likely to give one much of an understanding of how the system or the user will behave. This understanding is important to the human factors analyst who is analyzing the interface specification for design flaws. Questions such as "What happens when the user presses the mouse button here?" or "What mode is the system in when she exits from this level?" are not easily answered unless the components of the taxonomy are articulated together. Modeling the system behavior facilitates specification of the design and aids analysis of the interface by giving an overall view of how the final system will look and act. If one could model user behavior as well, then much of the human factors analysis could be performed without behavioral studies. However, since powerful models of human behavior do not exist, a detailed task analysis is typically all that is developed during the design of interfaces, while behavioral studies are relied upon to capture user behavior. Thus, the focus of modeling is on the system behavior alone. (Note however, that Kieras and Polson [1985] used the GOMS model of Card, Moran, and Newell [1981] as the basis for a production system simulating user behavior during a text editing task). The next section describes how the system can be modeled and how this model relates to the component taxonomy of the USI.

MODELING SYSTEM INTERFACE BEHAVIOR

A Finite State Model of System Behavior

Following the lead of others (Kieras and Polson,1985; Jacob,1983; and Parnas,1963), the interactive system is modeled as a finite state machine. Commonly this would be called a "black box" approach because there is no attempt to explain the

internal workings of the systems software and hardware; rather system behavior is defined only in terms of inputs from the users and output from the system.

A finite state machine, or finite automaton, is a computer-like device that has an input mechanism and an output mechanism. Its output is a function not only of the current input, but also the past history of inputs. At any given moment, the machine is in a particular state waiting for input that would cause it to transition to another state. In interactive systems, a state can be thought of as the system being in a certain *mode*. More precisely, to represent the behavior of the system interface, there must be a definition for a set of states, a set of inputs, a set of outputs, and a set of state transition functions that map combinations of previous states and the current input to a new state. (For a formal definition of finite state automa see Wulf et al., [1981]). A set of state transition functions defines what is called the *control flow* of the system. Therefore to define the system behavior one must specify the inputs (user actions), outputs (computer generated outputs such a graphics or sound), and control flow (when and where the the system changes to different modes).

Relating USI components and models of behavior. Models of user behavior and system behavior can be related back to the taxonomy of the user-system interface described earlier. Figure 1 shows how the components of the USI are allocated to different parts of the models. The model of user behavior is included for completeness even though it will not be developed here. There are three sections to the models: the user's conceptual component, the communication and physical components, and the system's conceptual component. Notice how the communication and physical components have two viewpoints: that of the user and that of the system. The user performs actions which are seen by the system as inputs, and the outputs from the system in turn affect the user's perception. The shaded gray area in the figure represents the parts of the model that need to be defined before an analytic human factors analysis of the interface design can be performed.

The conceptual component of the USI taxonomy is partially relevant to the system model, and only if one uses a broad definition of cognition. It is difficult to consider the task level to be appropriate to the realm of system behavior. One can imagine

technological change will make a limited definition of "task" relevant to system behavior. Semantics, on the other hand, are the legitimate domain of system behavior. Defining system functionality and operations is part of the semantic specification of the system.

The demands on a specification tool are such that the tool should describe system behavior while simultaneously articulating the details of each level of the USI component taxonomy. What set of tools are best suited for this task? To discover the range of available specification tools, a survey of USI specification tools was performed. The results of this survey were used in a subsequent engineering tradeoff analysis to find a small group of specification tools that can be used for human factors analysis. The next sections describe the survey and the tradeoff analysis.

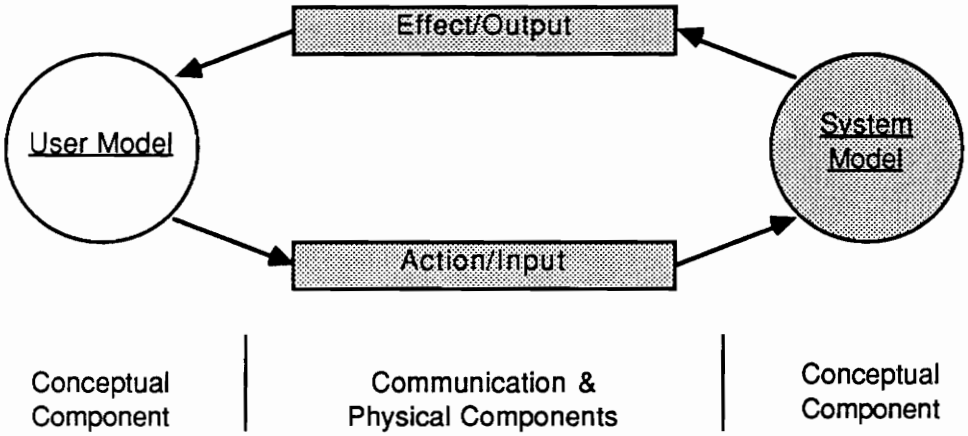


Figure 1. *The USI taxonomy components and their relation to the system model.*

CHAPTER 3. USER-SYSTEM INTERFACE SPECIFICATION TOOLS

A SURVEY OF SPECIFICATION TOOLS

The specification of USIs has been approached many different ways. The techniques have come from many diverse fields: computer science, linguistics, cognitive psychology, and system design. While some specifications are effective for the purposes they were originally designed, they have varying effectiveness as design tools.

System specification tools can be roughly categorized into four groups: semi-formal types, BNF-like (Backus Naur Form, [Backus, 1959]), programming logic, and transition network representations. The tools in the first three groups represent the work primarily of computer scientists and linguists, and they are formal and precise. Semi-formal specification tools are mostly the result of work in the human factors and system design fields.

Semi-Formal Representations

There is group of tools that are not formal enough to be used in formal proofs of system properties, however, they have several advantages over formal notations: (1) specifications using them are more easily understood and created; and (2) pragmatic aspects of the interface are better represented.

Rapid-prototyping. Prototyping is the construction of a very restricted representation of a full implementation of an interface, generally for the purpose of testing the feasibility and adequacy of the design. Frequently, computerized tools are provided to the interface designer to speed the process of prototype construction. The tools make prototyping *rapid*. By definition, the prototype is limited in functionality by

the lack of power of the prototyping tool, and the need to reduce the amount of time spent creating it. The purpose for constructing a prototype can be viewed as an attempt to specify the system. This is the perspective that this survey of specification tools takes. A rapid-prototyping tool is just another system specification tool.

The focus of past rapid-prototyping tools has been on representing command language or menu-driven interaction styles. The trend in software and hardware is toward concurrent direct manipulation dialogs that require highly graphical interaction styles. The designer is faced with a less defined and more difficult task than when designing command language style interfaces. There are a variety of graphical attributes that can be manipulated on modern workstations and their low cost has made them commonplace. A designer needs tools that allow quick and easy access to variations in the layout of visual attributes of a display. There are very few tools described in the literature that can prototype complex graphical interfaces.

Many rapid-prototyping tools use a high-level language generically called an Interface Definition Language to specify the structure and behavior of the prototype. These languages vary in syntax, but, generally they are very structured and and similar to Pascal. There are two major drawbacks with this style of prototyping. First, expertise in programming is required (something not all designers have); and second, a textual representation of a graphical interface has little compatibility with the designer's view of an interface.

Relatively recent developments in rapid-prototyping tools (e.g., Trillium [Henderson, 1986] , Peridot [Myers, and Buxton, 1986], Skylights [Skylight, 1987], and GIDS [Overmeyer, and Campbell, 1984]) have signaled the appearance of what one might call WYSIWYG (What You See Is What You Get) style tools. That is to say, a prototype is specified mainly interactively using predominantly direct manipulation techniques, and the results are displayed immediately. The designer gets to see the results of changes in the design in a form that resembles the final design. Peridot may be a glimpse into the future in rapid-prototyping tools. It is driven by a knowledge-based system that understands some properties of interactive objects. During a typical session, one would be constructing a pull-down style menu (like the Macintosh), and the

system would query the designer about whether certain attributes were desired such as reverse video, borders, title bars, etc. This advice occurs simultaneously to the interactive inputs from the interface designer and can be ignored if desired. With large knowledge bases, this technique in the future may facilitate the construction of very complex graphical interfaces.

Those rapid-prototyping tools that use Interface Definition Languages and cannot represent graphical interfaces were not considered candidate system specification tools for the engineering tradeoff analysis. They offer little advantage over other structured specification tools such as programming languages that are discussed later.

Storyboards. Early in the design stage, sketches or drawings of the system interface are frequently called storyboards. The level of detail that is described by these drawings can vary. They can be augmented by a written scenario that logically connects the drawings according to certain user inputs and system states. If this is the case, then a storyboard is really a pencil and paper prototype. Although storyboards of a design seems too primitive to be used in a human factors analysis, many usability problems arise solely from the organization of the display. These problems can be found in a storyboard during an analytic analysis.

Semi-Formal Grammar. In contrast to formal notations, the semi-formal grammar (SFG) of Monk and Dix (1987) is aimed at the designer not specialized in formal grammar. Overall, the concept is to reduce system behavior to descriptions of users actions and system effects. This is the "black box" view of the system without much information about states of the machine. The procedure for creating a SFG representation is:

1. Identify the major display contexts of the interface. Describe these in a very general way.
2. Next specify the "action" vocabulary. This is a listing all possible lexemes a user can input.
3. For each display context, write action-effect rules for each possible input in that context. These are short textual descriptions of system behavior based on a particular user action.

Figure 2 is an example of one display context in a simple text editor.

The main advantages of this specification technique is the relatively ease with which one can construct a system representation that both end users and designers can understand. However, since little structural information is given by the grammar, it can be difficult to tell how the system is organized. Clearly, very little information about the physical attributes of the system can be explained textually.

BNF Grammars

Several specification tools are based on the BNF grammar formalism. Reisner (1981,1984) uses BNF to describe the "action language" of the user-computer dialog. This language is composed of user inputs and actions. Reisner (1981) defines a BNF specification as a production rule grammar:

> A production rule grammar describes a language as a set of rules for describing correct "strings" (e.g. sentences) in a language. Any particular string can be described by the particular rules involved in producing it (p. 231).

Some examples of the notation will serve better to explain its nuances. Figure 3 provides three examples taken from Reisner (1981) in which she describes parts of a non-trivial graphical editor using a BNF notation. (In the paper she fully specifies the graphical editor in terms of the notation). Following each BNF example in Figure 3 is an English description as if one were reading the grammar out loud.

One can see the use of common metasymbols in this BNF grammar: + ("and"), | ("or"), ::= ("is composed of"). In general, all BNF-like grammars are characterized by:

1. a set of terminal symbols (the words in a language, e.g., CURSOR IN RED from figure 3);
2. a set of non-terminal symbols, to show the structure of the language (e.g., "colored shape");
3. a starting symbol;
4. the metasymbols "+", "|", etc;
5. and rules constructed from the items above.

By using this notation, one can describe the sequence of user actions (i.e., button pushes, joystick motions and mouse movements) used to interact with the system. The emphasis of the notation is on specifying the syntax of user communication and system inputs, and not on specifying system behavior and outputs. Little information about semantics and system functionality is defined. From a BNF grammar specification of a system it is difficult to understand system behavior.

In an attempt to represent the computer responses to the user actions in a BNF style, Shniederman (1982) added a notational convention to distinguish whether a user input occurred or when a computer output occurred. This notation is referred to as a Multi-party grammar. The member of the party that is in the process of communicating is distinguished by labeling non-terminals with either an H: (human) or a C: (computer). Shniederman added two additional features to the standard BNF notation:

- assignment of values to non-terminals and the use of square brackets to output the values, giving the notation the ability to store information about states of the system.
- a special non-terminal that is a "catch-all" if no other rule can be satisfied, insuring that the system can always complete a production successfully.

After these significant additions, the notation can be mapped to a transition network representation to be described in the next section. A major drawback of using the BNF notations is their technical nature, making specifications difficult to generate and interpret.

C1. Welcome banner and prompt for document name.
C2. Task menu (edit/create, spelling check, print, etc.)
C3. Command mode (page with "Enter command" under a line of dashes at the bottom).
C4. Insert mode (page with no line of dashes and "Insert in progress" at the bottom).
C5. to C19. Other command modes

Major display contexts for an example text editor.

A = {A, ... Z, a, ... z, 0, ... 9, !, ", #, %, &, (,...} (Printable characters)

D = {1,2,3,4,5,9} (Menu digits)

C = {A,a,B,b,C,c,D,d,F,f,H,h,I,i,J,j,K,k,L,l,M,m,
N,n,O,o,P,p,Q,q,R,r,S,s,T,t,V,v,X,x,Z,z} (Commands,
D & C are a subset of A).

<space>
<tab>
<del>
<back space>
<return>
arrow_keys = {<up>, <down>, <left>, <right>}
<esc>

Action vocabulary for the example system.

R1. **A**::Character appears on the screen to the left of cursor begining of next line, if the edge of the screen is reached. Letters appear in upper case.

R2. <del>::Character to the left of the cursor disappears, unless none typed.
R3. <space>, <back space>, or arrow_key::cursor moves appropriately
R4. <tab>::inserts 8 spaces
R5. <esc>::no effect.
R6. <return>::if only letters and numerals have been typed changes mode to C2. otherwise, beep, and display message, "error in document name", pause a short while and redraw screen as it was initially in C1.

Action-Effect rules for C1 (Welcome banner and prompt for document name).

Figure 2. *Example of a Semi-Formal Grammar (adapted from Monk and Dix, 1987)*

new color ::= CURSOR IN RED | CURSOR I BLUE | CURSOR IN GREEN | ...
"A new color is selected when the user dips the cursor into one of the colors."

colored shape ::= color + shape | shape + color
:"A colored shape consists of either a color followed by a shape or
a shape followed by a color."

select-d-shape ::= select separate d-shape + describe separate d-shape
"To create a discrete shape, the user has to select it and describe it."

Figure 3. *Three examples of the BNF grammar (adapted from Reisner, 1981)*

Programming Languages

High level programming languages have been used to specify computer interfaces formally. The strength of this approach is allowing formal and uniform specifications of the semantic and syntactic levels of the interface. Roach and Nickson (1983) suggested the use of Prolog (a logic programming language) to first define the input and output details of the interface and then specify the control structure of the inputs and outputs. They used Prolog to specify the interface for an air traffic control system.

Foley (1987) developed his own Pascal-like high level interface specification language. His language only specifies the semantic level of the interface. An advantage to this is that a computer can be used to generate design alternatives automatically based on the initial specification. Once a set of semantic objects and operations is specified, the computer is used to generate many different possible syntax specifications.

Programming language specifications have the advantage that they can be formally analyzed and transformed by computers. The disadvantage is that they it require programming expertise.

Transition Networks

A transition network diagram is a graphical representation of a finite state machine. A simple diagram has a set of nodes representing terminal states. These nodes are connected by directed arcs that represent transitions to other states. An arc is transitioned when specific input and state conditions are met. Using these diagrams to represent interactive systems was first proposed by Parnas (1969). He suggested that the diagrams are useful during the design stages to discover inconsistent designs that the user would find difficult to use. Jacob (1983) also advocated using transition diagrams to represent interactive systems. He adds the concept of nesting diagrams to minimize the level of detail available. The system can then be "opened up" by going down levels of detail. Some examples of transition network diagrams will serve to illustrate nesting.

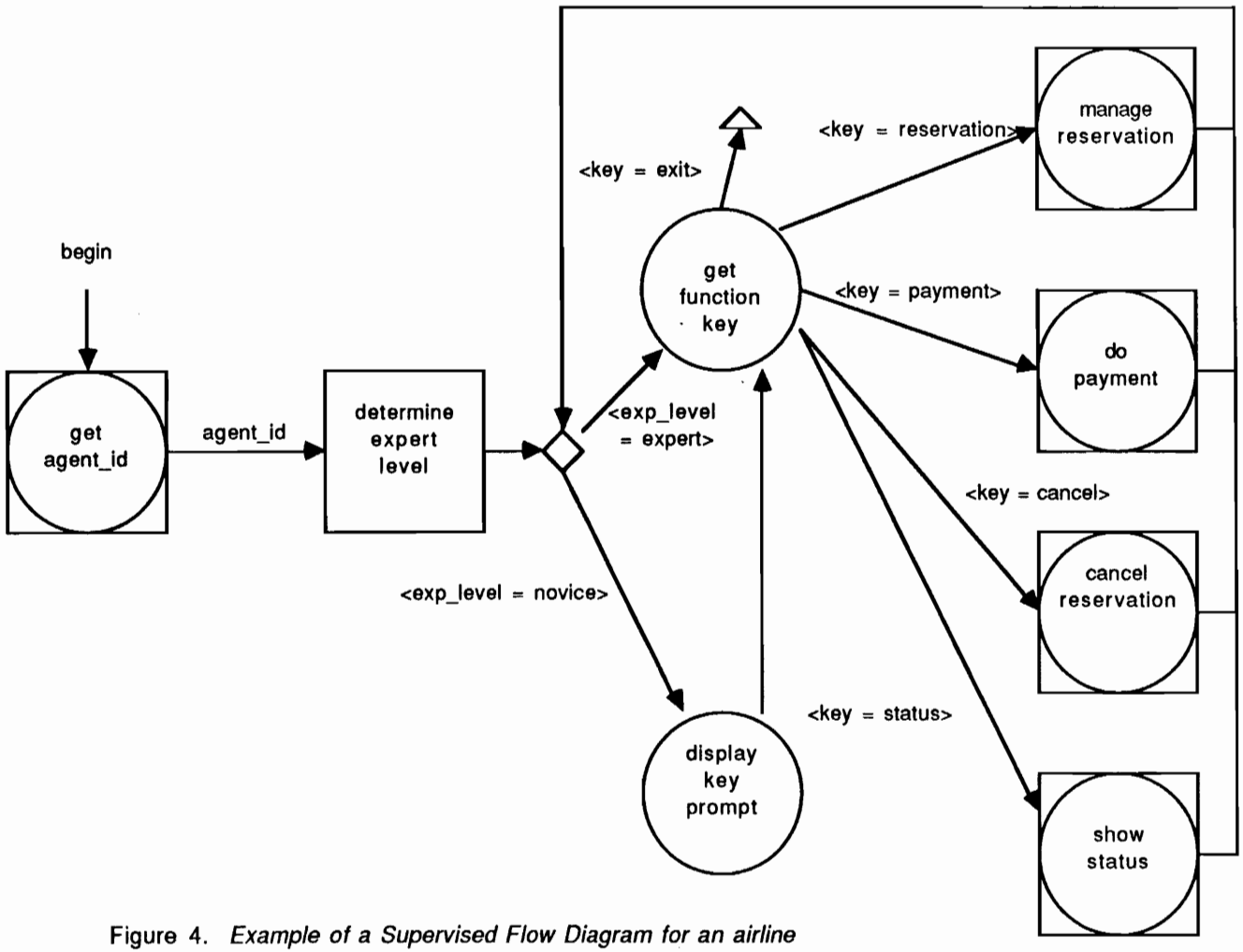


Figure 4. *Example of a Supervised Flow Diagram for an airline reservation system (adapted from Hartson et al., 1986).*

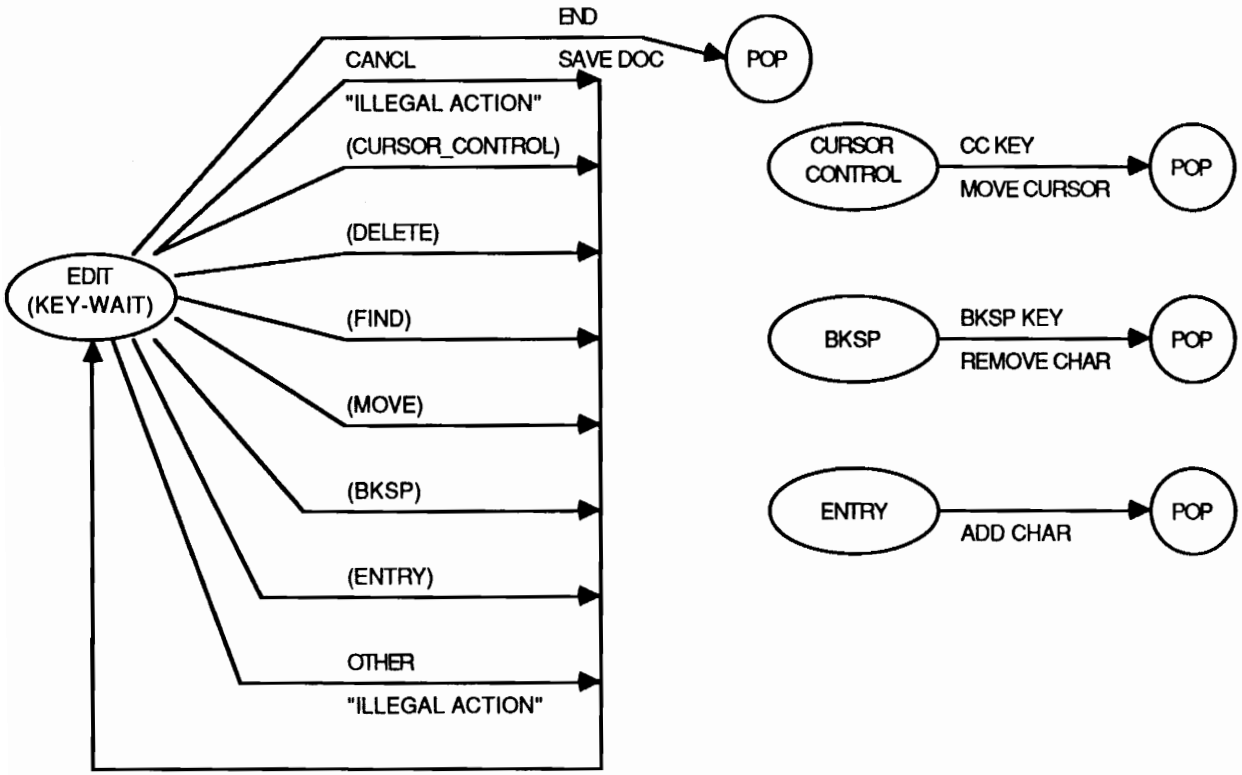


Figure 5. *Example of a Generalized Transition Network (adapted from Kieras and Polson, 1985)*

Supervisory Flow Diagram (SFD). This notation is a modified form of the simple transition network diagram. The notation was developed as the basis for an User Interface Management System and seen as a way for software and human factors engineers to communicate about the evolving design (Yunten and Hartson, 1985). Figure 4 is an example of how this diagram is used to represent a portion of an airline reservation system. Basically, the system has the agent log onto the system and perform several reservation operations for customers. Circles in the diagram represent states where user input is required. A square represents some computations that must be done by the computer before the next state is reached. Combinations of circles and squares represent states where both user input and computation take place. One can see that nesting is used in this diagram by considering for example the "manage reservation" state in the diagram. It is clear that this state would have to be expanded and the details specified of what it means to the user and the computer to "manage" a reservation.

Generalized Transition Network (GTN). The most developed of the transition network diagrams is the Generalized Transition Networks (GTNs) of Kieras and Polson (1985). These diagrams include not only nesting of states, but nesting in condition/action pairs on the directed arcs. The advantage of this method is that it's not necessary to represent components of the diagram that are used over and over again. This is akin to the subroutine features in many computer programming languages.

From diagrams such as Figure 5 it becomes clear what a sequence of inputs means to the behavior of the system. This example specifies a portion of a text editor. The diagram demonstrates how nesting of state transitions is represented. If a name on the diagram is enclosed by parentheses, then there is a separate diagram of this name defined in more detail elsewhere. Names that are above the arc represent the condition that must be satisfied before the arc is transitioned. Names that are below the arc represent the action that is taken when the arc is transitioned. Some arcs can be transitioned without meeting any conditions , while others have very complex transitions functions. In the example the arc labeled "(BKSP)" is a nested arc condition. This condition is defined fully to the right in the small diagram that has a state labeled "BKSP". Because of features like this, the hierarchical and multi-mode nature of interactive systems can

be represented. A weakness of the notation is that only a paucity of information is given about the form and content of user inputs and system outputs.

AN ENGINEERING TRADEOFF ANALYSIS OF SPECIFICATION TOOLS

Three different analyses were performed. First the specification tools were examined as to their ability to represent levels in the USI taxonomy. This analysis was not powerful enough to be able to select the best specification tools. A second analysis was performed to look at the specification tool's ability to represent the components of the system behavior model. It was found that this analysis did not give a complete picture of the characteristics of the specification tools. Specifically, the pragmatic characteristics that would determine the usefulness of a specification tool to the interface designer. The third and last analysis used a set of criteria developed to select those tools that are usable by the human factors analyst and the interface designer.

Analysis of Specification Tools and the USI Taxonomy

In this analysis the question was asked about each specification tool, "to what degree can this system specification tool represent a level in the USI taxonomy?" Table 1 shows how this question was answered for each level of the USI taxonomy and each specification tool. A score of *limited* means that relative to the other specification tools the tool has some ability to represent the level of the taxonomy. A score of *yes* indicates a tool is the best or as good the others at representing the properties of the taxonomy at a particular level. A blank score indicates that the tool has basically no method of representing a level in the taxonomy.

As an example of how the analysis was performed, the analysis of the semi-formal grammar (SFG) will be explained. The first level in the taxonomy is the task level. None of the specification tools have the capability to represent user tasks by default because they are all system specification tools. A score of "limited" was given at the semantic level because the SFG tool has some ability to specify the meaning of a user input as it relates to the functionality of the system. As part of creating a SFG specification one must define the entire input vocabulary. This constitutes in large part

the syntax of allowable user inputs. Therefore the syntactic level is defined by the SFG specification. At the lexical level, the SFG does represent the lexemes used for input. Clearly at the spatial layout, auditory layout and device levels, a SFG specification will be very weak since one is limited to only textual descriptions of visual, auditory, and gestural attributes of the system.

From the table, it can be seen that there are many gaps in the capabilities of specification tools especially at the spatial, auditory, and device levels. This points to a need to develop new specification tools to fill the gaps. The analysis at this level is not detailed enough to declare logically a set of specifications tools as the best for specifying the system. The next analysis takes the view that what really needs to be specified by a specification tool is the behavior of the system. A tool, or set of tools, needs to be able to define the input, output, and control flow aspects of the system.

Specification Tools and the System Model

This analysis looks at the abilities of system specification tools to represent the parts of the system behavior model. If one goes back and looks at Figure 1, it can be seen that any complete input and output specification would have to represent simultaneously the levels contained in the physical and communication components of the USI. This observation led to development of the following analysis procedure:

1. For Input, a specification was scored highly, if, relative to other specifications, it more completely represented the communication and physical components of the USI at all five of its levels.
2. For Output, a specification was scored highly, if, relative to other specifications, it more completely represented the communication and physical components of the USI at all five of its levels.
3. For Control Flow, a specification was scored highly on its ability to describe precisely the mode changes and state transitions of the system based on previous inputs.
4. Scoring of a specification tool was made on a scale of 1 to 5. Whole numbers only are allowed.
5. The scoring was made relative to the other tools represented in the survey.

Table 2 summarizes the results of the analysis. As an example of how the analysis was performed, the BNF tool will be examined. Why is this tool ranked last? The only good score (3) that the BNF specification tool received was on its ability to represent inputs to the system. The reason for this is that the survey of specification tools described the BNF as defining the user actions that input to the computer system. Since this is the tool's purpose, it is scored highly on input. However, there is no mechanism for this tool to represent system output directly. Thus it was scored the lowest possible (1) in this category. Without any information about outputs, the BNF specification can only weakly define the system's control flow.

The rapid-prototype comes out as the clear winner based on this analysis. However, this analysis does not provide the level of information needed to determine if a specification tool will be useful for an analytic human factors analysis of an interface. Which specification tools are really usable by the human factors analyst? The next section describes a set of criteria developed to select system specification tools that represent the system behavior and are also usable by the person performing the analysis.

Analysis Using Pragmatic Selection Criteria

Criteria used to select specification tools. Most system representation techniques have their source in the computer science field where proving formal properties such as "completeness" and "Turing equivalence" are important. Since the goal is rarely formal proof when designing and analyzing the USI, many of the techniques may not be relevant to the needs of the interface designer and human factors engineer. The purpose of this section is to develop a set of criteria that can be used to select system specification tools that are helpful to the interface designer and evaluator. It is instructive to look at the criteria used by other investigators to select a tool for specifying an interactive system.

TABLE 1

Analysis of Specification Tools Using the USI Taxonomy

	Task	*Semantic*	*Syntactic*	*Lexical*	*Spatial Layout*	*Auditory Layout*	*Device*
Rapid-Prototype		limited	yes	yes	yes	limited	limited
Semi-Formal Grammar		limited	yes	yes	limited		
Storyboard		limited	yes	limited	yes		
BNF			yes	limited			
Multi-Party Grammar			yes	limited			
Programming Languages		yes	yes				
GTN		yes	yes				
SFD		yes	yes				

TABLE 2

Analysis of Specification Tools Using the System Model

	Input	*Output*	*Control Flow*	*Totals*	*Rank*
Rapid-Prototype	3	5	3	11	1
Semi-Formal Grammar	3	2	3	8	5
Storyboard	2	4	2	8	5
BNF	3	1	2	6	8
Multi-Party Grammar	3	2	3	8	5
Programming Languages	1	2	4	7	7
GTN	2	2	5	9	2.5
SFD	2	2	5	9	2.5

Kieras and Polson (1985), in their effort to represent the system behavior such that the interaction between user and system could be characterized in a simulation model, identified a set of six properties of any representation. It should:

- have well defined formal properties, so that its correctness can be explicitly determined;
- make interactive systems easy to represent;
- be modular, so that as much or as little of the system can be represented as desired;
- easily represent hierarchical control or structural relations;
- not be committed to any particular hardware of software implementation;
- be easy to represent the features of the system that have psychological implications.

Based on these properties, they selected and developed a transition network representation (GTN) over BNF notations. They say that BNF notations do not clearly indicate the hierarchical structure of the system and that system semantics are not well represented.

Jacob (1983) in his survey of interface specification techniques, also selected a transition network representation. The properties he used to select a tool were:

- the specification should be easy to understand;
- the specification should be precise;
- it should be easy to check for consistency in the design;
- it must be powerful enough to express non-trivial systems;
- it should separate function from implementation;
- it should allow a prototype to be constructed directly from it;
- the structure of the specification should be closely related to the user's model of the the system.

It can be seen that the surveys use similar selection criteria although the definitions of these criteria are not always clearly defined. The analysis required that a

new set of criterion be defined. Incorporating some of the criterion from the work described above and addressing the pragmatic concerns of the human factors analyst and interface designer, the following four criterion were derived:

Understandability. The specification must be easily understood by the interface designer without requiring her to have in depth knowledge of computer science or linguistics. Ideally, the specification can be interpreted by a customer or a subject matter expert, to such a degree that they can make design recommendations.

Efficiency. The time and effort to specify the system using the specification tool must be less than that needed to produce the target application's software. It is conceivable that a specification may give information about design flaws that could not be discovered with an analysis of the final system. In this case, it may be worth the effort to specify the system before implementation even if it takes more time than to write the software. This situation is unlikely to occur often and is not considered in the specification tool selection process performed here.

Expressive Power. The specification tool must be powerful enough to represent non-trivial system behavior in an abstract and precise way. Given that there are many interaction and interface styles, the ability to represent a system at high levels of abstraction is desirable when finding design flaws that are not the result of low-level details of implementation. Details of implementation can hide the fact that there are abstract classes of design flaws and system behaviors that cannot be analyzed at lower levels of detail.

Fidelity. The specification tool should be able to describe the look and feel of the final system as much as possible. There are many interaction and dialog styles, (e.g., form-filling, command language, menu-driven, direct manipulation, and speech I/O) each with its own unique characteristics that need to be represented accurately.

The above criteria selected to evaluate the specification tools may not be the necessary and sufficient criteria to predicate the successful usage of the tool by a human factors engineer conducting an analytic analysis of a USI. Both criterion deficiency and criterion contamination must be considered (Blum and Naylor, 1968). Criterion deficiency represents the extent to which the *actual* criterion fail to predict certain variance necessary to the *ultimate* criterion (the theoretical "true" criterion). Conversely, criterion contamination is the variance in the *actual* criterion unrelated to the variance in the *ultimate* criterion. Criterion contamination includes both bias and error. To the degree that either criterion contamination or criterion deficiency exits, the results of this analysis of specification tools are distorted.

Analysis Using the Criteria

The specification tools were scored on each of the four pragmatic criteria for each of the four pragmatic criterion for each of the three components of the the USI represented in the finite state model: input, output, and control flow. The scoring was based on extensive knowledge of the specification tools. The scores were reviewed by two other human factors engineers somewhat less familiar with the specification tools. Their comments resulted in minor changes in the scores. The scores were then summed across the criterion to yield a single composite score. Equal weightings were used on each of the criterion.

Alternatively, the multiple criteria might have been treated separately. Experts on criterion development have not agreed on the relative merits of composite or multiple criteria (Blum and Naylor, 1968). Each type has a difficulty associated with it. The difficulty with multiple criteria is in trying to view the criteria independently while needing to predict the value of each specification tool. One solution would be to order the criteria based upon importance and select a minimum level for each. Then each specification tool would be subjected to a series of tests against these minimum levels. Those tools that passed all the tests would be selected as being best for analytic human factors analysis. This process is know as a multiple hurdle system. However, selecting appropriate minimum levels and an order for the examining the criteria is analogous to the difficulty in determining weightings for a composite criterion.

When creating a composite criterion based on sub-criteria, the problem basically reduces to the difficulty of developing weightings for the sub-criteria that can be combined. Several methods have been suggested to generate weightings, such as: "expert" judgement, proportional reliabilities, proportional correlation with other variables, predictability, factor analysis, and maximal differences. In the absence of strong evidence to support the validity of unequal weightings, using an equal weighting scheme probably yields less overall error in the long term.

The following procedure was used to select specification tools according to the pragmatic criteria.

1. A high score indicated that a specification most closely conformed to the definition given for a criterion as compared to the other specification tools. This judgement was based on experiences with a particular tool as provided in the literature.
2. Scoring was done on a scale of 1 to 5. Only whole numbers were allowed.
3. The scoring was made relative to the other tools represented in the survey.

The results of the analysis based on the criteria and the input component of the system model are shown in Table 3. Notice how the rank of the BNF specification is relatively low. In an earlier analysis summarized in Table 2 the BNF specification was ranked relatively high in its ability to represent input. This apparent discrepancy can be explained by realizing that the sheer ability (expressive power) of a specification tool is only 1/4 of the criteria. A BNF specification is difficult to interpret and requires computer science training which most human factors specialists do not have. Thus, it is the understandability and efficiency scores that are low. On the other hand, the semi-formal grammar (SFG) representation of input is relatively easy to construct but is not very powerful. Therefore, the SFG specification tool receives the highest overall rank to represent input.

The results of the analysis relative to output are given in Table 5. Rapid prototyping comes out way ahead of the other specification tools in representing output. Even though it takes relatively long to specify a prototype using a computer-based design tool, the fidelity and understandability of the resulting specification receive high scores.

Specifications of control flow come in two forms. One way is that the specification implicitly contains a control flow specification. This is the way the control flow is specified by a rapid-prototype. By observing the sequence on inputs and outputs and seeing how the system behaves, control flow can be inferred. A better method is to represent control flow abstractly through the notations of the specification tool. Transition network diagrams are an example of this type of specification tool. As shown in Table 5, the Generalized Transition Network (GTN) specification tool is ranked as the

best tool to represent control flow. The Supervisory Flow Diagram (SFD), though not any less capable of representing control flow, uses symbols and notations that are not familiar to the human factors analyst and make the tool less usable. Therefore, two semi-formal methods which only indirectly express control flow, are ranked just as high by the composite criterion.

Table 6, summarizes all of the results of the analyses by combining all the components of the USI model into one score for each of the pragmatic criteria. It is interesting to note that the semi-formal specification tools all ranked very highly relative to the formal specification tools.

TABLE 3

Analysis of Tools for Input Specification

	Understandability	*Efficiency*	*Expressive Power*	*Fidelity*	*Totals*	*Ranks*
Rapid-Prototype	4	3	2	3	12	2
Semi-Formal Grammar	4	5	2	2	13	1
Storyboard	3	4	1	1	9	5.5
BNF	2	2	4	1	9	5.5
Multi-Party Grammar	2	2	4	1	9	5.5
Programming Languages	1	2	3	1	7	8
GTN	2	4	2	2	10	3
SFD	2	3	2	2	9	5.5

TABLE 4

Analysis of Tools for Output Specification

	Understandability	*Efficiency*	*Expressive Power*	*Fidelity*	*Totals*	*Ranks*
Rapid-Prototype	5	3	4	5	17	1
Semi-Formal Grammar	4	4	2	2	12	3
Storyboard	4	4	2	4	14	2
BNF	1	1	1	1	4	8
Multi-Party Grammar	2	2	4	1	9	5.5
Programming Languages	2	2	4	1	9	5.5
GTN	3	3	2	2	10	4
SFD	2	2	2	2	8	7

TABLE 5

Analysis of Tools for Control Flow Specification

	Understandability	*Efficiency*	*Expressive Power*	*Fidelity*	*Totals*	*Ranks*
Rapid-Prototype	4	3	2	3	12	3.5
Semi-Formal Grammar	3	4	3	2	12	3.5
Storyboard	4	4	2	2	12	3.5
BNF	1	1	1	1	4	8
Multi-Party Grammar	2	2	2	2	8	7
Programming Languages	2	2	4	2	10	6
GTN	4	3	5	2	14	1
SFD	3	2	5	2	12	3.5

TABLE 6

Analysis of Specification Tools Across Input, Output and Control Flow

	Understandability	*Efficiency*	*Expressive Power*	*Fidelity*	*Totals*	*Ranks*
Rapid-Prototype	13	9	8	11	41	1
Semi-Formal Grammar	11	13	7	6	37	2
Storyboard	11	12	5	7	35	3
BNF	4	4	6	3	17	8
Multi-Party Grammar	6	6	10	4	26	6.5
Programming Languages	5	6	11	4	26	6.5
GTN	9	10	9	6	34	4
SFD	7	7	9	6	29	5

Conclusions

The set of system specification tools described in the specification tool survey have been reduced to three by the analysis. They are: the Semi-Formal Grammar for input specification; the rapid-prototype for output specification; and the Generalized Transition Network for control flow specifications. Overall, as shown in Table 6, the rapid-prototype is the preferred USI specification tool, followed by the Semi-Formal Grammar and storyboards. If one accepts the notion that in many ways a storyboard is a very simple rapid-prototype, then the Generalized Transition Network is really the third highest ranked specification tool overall. If one wanted to emphasize one or more of the criteria based on their own special requirements, unequal weightings could be assigned and the rankings recalculated. In this way the raw data in the tables can be used to come up with a set of specification tools tailored to the requirements of a particular design environment.

The engineering tradeoff analysis has indicated that USI specification tools differ greatly in their ability to represent the input, output and control flow components of an interface. In an area where one specification tool is weak, another is strong. If this is true, the quality of the human factors analysis of an interface will suffer if only one USI specification tools is used. The analysis suggests the need to use a set of specification tools. Further, it is logical to conclude that design defects will be found most easily by the human factors engineer when the type of design defect (a defect in input, output, or control flow) matches the representational strengths of the specification tool. The next section describes a behavioral study that attempted to validate this conclusion of the engineering tradeoff analysis with actual user performance.

CHAPTER 4. BEHAVIORAL STUDY

OBJECTIVES

Given a set of design defects in an interface that are evenly divided between defects in its input, output and control flow components, one would expect the performance of the human factors engineer in finding these defects to vary according to which specification tool was chosen. In particular, the performance would vary in a systematic way according to the strengths and weaknesses of a specification tool in representing input, output and control flow. The purpose of the behavioral study was to demonstrate that the choice of a specification tool affects the performance of a human factors engineer, and that the affect is predicted partly by the engineering tradeoff analysis.

Participants in the study were asked to find design defects that were systematically imbedded into an interface that was specified in one of three ways: as a semi-formal grammar (SFG), a rapid-prototype, or as a generalized transition network (GTN). The selection of these three specification tools reflects their rankings from the engineering analysis as the best candidates for representing input,output, and control flow respectively. The hypothesis was that the participant's performance at finding design defects related to input would be best when using a SFG. Performance at finding defects in output would be best when analyzing a rapid-prototype specification. And that finally, their performance at finding control flow design defects would be best when a GTN was used. Performance was measured by such indicators as: the percentage of design defects of a particular class that were found; how long it took to compete the analysis; and the number of false-alarms generated. The next section describes how the interface specifications were created and how a set of design defects was generated.

METHOD

The Task Environment

The interface and task scenario. The interface was based on a RAMCAD (Reliability and Maintainability CAD) task. RAMCAD is a design environment being jointly developed by the Virginia Tech Industrial Engineering & Operations Research department and TRW that attempts to integrate reliability, maintainability and other engineering and economic analysis tools into the CAD process. Computer-aided design tasks are rich in inputs, graphical outputs and complex changes of context and control, making RAMCAD a good source of interface concepts that were used to formulate a simplified interface design that was specified with each of the specification tools. The RAMCAD system will be a menu-driven multi-tasking interface with windowing capabilities typical of many currently available CAD environments. Both keyboard and locator (e.g., mouse, graphics pad, etc.) inputs will be accepted by the system.

For the study, a small, typical, RAMCAD task was extracted from the proposed task scenario that a designer using a RAMCAD system is envisioned to perform. The task was to define part of an electrical circuit in a larger circuit assembly, then perform a thermal analysis on this circuit and see how reliability measures were effected. Appendix A contains a copy of the the task scenario. It is written from the point of view of a designer using the RAMCAD system. Participants in the study. referred to this scenario during their analysis of one of the USI specifications.

Construction of interface specifications. The next step was to specify the interface as a rapid-prototype, a SFG, and a GTN. At first, the USI specifications were created without any design defects. The rapid-prototype specification was built on a personal computer using a commercially available prototyping tool (Skylight, 1987). Although the RAMCAD system will ultimately run on a much more powerful engineering workstation, it was possible to simulate much of the look and feel of a CAD system. The prototype used a mouse for locator input. Keyboard inputs were no accepted by the prototype. To allow both kinds of input would have required extensive and time consuming modifications to the rapid-prototyping tool. This limitation on input

modality was felt to be a realistic example of the reduced functionality inherent in rapid-prototyping tools.

The SFG was written on a text editor and the GTN was created using a computerized drawing tool. The time to construct each of the specifications was recorded to give some idea as to the efficiency of each specification tool. The absolute times are subject to learning and ordering effects since the specifications were not constructed simultaneously, and variations in the quality of the tools used to construct the specifications, however, the order of the specifications is likely correct. It took 48 working hours to construct the rapid-prototype, 24 hours to construct the GTN specification, and 20 hours to construct the SFG specification. It was found that rapid-prototyping was not very rapid and that the demands of the GTN formalism required very careful attention to detail. This practical experience supports the results of the engineering tradeoff analysis summarized in Table 6 which shows the same ordering along the *efficiency* criterion.

Selection and generation of embedded design defects. Before design defects were generated the principle of *consistency* was selected from the interface design, and design guidelines literature (e.g., Williges, Williges, and Elkerton, 1987), as a principle recognized as important to a good design and one that could be systematically and easily violated in the interface environment. This principle means loosely that the similar contexts of an interface should be implemented so that the user can expect the interface to behave and appear in uniform ways. A systematic attempt was made to violate this principle in the interface. Five defects were created for each component of the interface (i.e., input, output, and control flow) for a total of 15 design defects. To ensure that a generated design defect corresponded to the violation of the design principle, a human factors professional experienced in interface analysis judged the prospective defects in the design.

The procedure that was used to generate design defects is:

1. Define the *consistency* principle as completely and clearly as possible.
2. Generate or revise the set of design defects.

3. The human factors expert will be asked to determine how the design principle was violated for each design defect and what component of the interface it involves. This will be done without coaching or comments in order to encourage independent agreement on design defects and reduce bias. The expert will then hear my reasoning as to the selection of a design defect.

4. Repeat step two and three. This process will continue until there is complete consensus on the definition of the design principle and how its violations are manifested in the interface design. If consensus cannot be reached after three iterations the current set of detects will be used throughout the study.

Some examples of defects generated for the study will serve to illustrate what constituted a design defect. The simplified RAMCAD interface was a menu driven system. Sub-menus appeared when one of the main menu item was picked using the mouse. A quick depress and release of the mouse button on a menu item represented a pick for the majority of the menu items. However, one menu item behaved differently in that the sub-menu disappeared unless the mouse button was continuously depressed. This is called a pull-down style menu. This example describes a design defect in input to the interface. It is an inconsistency because a different method is used to select a menu item similar contexts.

One design defect in output of the interface was that menu locations were not consistent in that they appeared in several different locations on the screen in similar parts of the interface.

As a final example, a control flow design defect was present in the interface such that if the user picked a type of reliability analysis from the reliability menu, the menu remained open. In all other menus in the interface, when a menu item was picked, the menu disappeared. This is another example of inconsistent design in the interface.

Two human factors specialists arrived at a set of fifteen design defects after several careful reviews. Table 7 shows the distribution of the design defects across the specification tools and USI components. Notice that none of the specifications contain all of the design defects. This was mainly due to inherent weaknesses in the specification tools. The GTN specification, for example, contains none of the output design defects. This points out the trouble of specifying output aspects of an interface using a GTN specification. The rapid-prototype specification contained two of the five input defects.

This was because the prototyping tool could only handle mouse inputs, while some of the input defects were tied to keyboard inputs.

It is interesting to see how design inconsistency were represented by the tools. In the rapid-prototype specification, inconsistencies were evident in the appearance of the screens, in the methods of input from the user, and in unusual system behaviors. Since the SFG specification is textual, inconsistencies showed up commonly as inconsistent wording in similar parts or blocks of text. The GTN specification's graphical nature made graphical inconsistencies in similar graphic forms the easiest way to search for possible design defect. These observations led to the development of preferred search strategies based on the type of specification tool being used. These strategies were taught to the participants during the study.

Appendix H contains the instructions given to the participants telling them how to run the rapid-prototype. Appendix I contains the SFG specification and appendix J contains the GTN specification used in the study. Notice how the nature of the specification tool affects the representation of the design inconsistencies described in appendix B.

Participants

Twelve graduate level human factors students were used in the study. They all had been in the graduate program at least one year, and were paid 5 dollars an hour for their time. The human factors specialist used in the selecting of design defects was a PhD. student in human factors that had research experience involving the study of human-computer interaction. This specialist was not paid for his time.

Apparatus

An IBM-PC AT running the SKYLIGHTS (Skylight, 1987) rapid-prototyping tool in the EGA, 16 color, hi-resolution(640x350), mode was used for the rapid-prototype specification. A Microsoft Bus Mouse (version 6.1) was used for input. The two other specifications were implemented on paper. A VHS format video tape recorder and camera was used to collect data during the rapid-prototype analysis sessions.

TABLE 7

Distribution of Imbedded Design Defects in the USI Specifications

	Input					Output					Control Flow				
	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
Rapid Prototype		•	•			•	•	•	•	•	•	•	•	•	
SFG	•	•	•		•		•			•	•	•	•	•	•
GTN	•	•	•	•	•						•	•	•	•	•

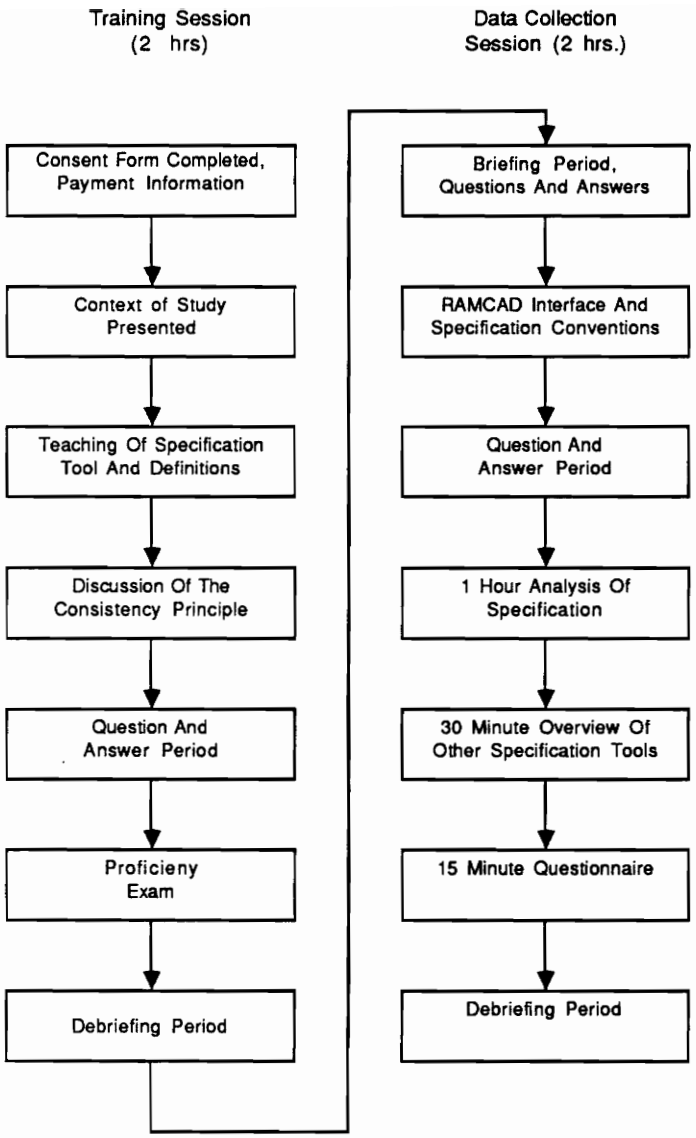


Figure 6. *Experimental procedure.*

Experimental Design

The experiment was a mixed-factors design. The between-subjects factor was the type of interface specification tool and had three levels (rapid-prototype, SFG, or GTN). The within-subjects factor was the class of design defect (input, output, or control flow). A subject analyzed the interface using one interface specification that contained as many of the fifteen design defects that could be represented by the specification tool.

Procedure

The temporal order of the experiment is shown in Figure 6. Each of the three groups followed this experimental process. The study is divided into a training session and a data collection session each lasting at most two hours. The goal of the training session was to give the participants a comprehensive set of skills that would allow them to focus on finding defects in the specification and not be consumed by the effort needed to interpret the specification tool. For all groups the training and data collection sessions were conducted by the author.

Training session. The participants in the SFG and GTN groups had no experience with their respective tools and needed longer training than the rapid-prototype group. During pre-testing sessions it was found that very little training on the prototype was required. Running the prototype was a familiar task to the participants. This was likely due to the extensive computing background of the participants. It was decided that the training and data collection sessions for the rapid-prototype group could be combined and performed on the same day. The SFG and GTN groups were trained in a classroom type environment. All participants in these groups attended their respective training sessions and returned two days later for the data collection session. Each participant in the rapid-prototype group was handled individually because of the lack of additional computing and video equipment needed for group training and data sessions.

During training the participants were allowed to take notes and ask questions. Several handouts were given that described: (1) the USI specification tool, (2) the consistency principle of design, (3) analysis techniques, and (4) common notations and

computer terms (see Appendice C, D and E for examples of the training materials). Participants were told that any materials they were given or had created themselves could be used during the data collection session. Deception was never the goal and participants were told in general terms that the purpose of the study was to evaluate the effectiveness of software interface specification tools. To set the context of the study, participants were asked to think of themselves as new employees at a computer firm. And that at this firm, interface designs were specified using a special tool. Thus the training session could be seen as a way for new employees to learn about the specification tool.

The first part of training session involved teaching the participants the specification tool they were to use during the analysis session. An example specification of a text editor was used to illustrate aspects of the notation and style of the specification tool. Some of the notational conventions that were peculiar to the RAMCAD interface were explained at this point in training. It was necessary that they had the skills to interpret the interface specification they would see during the analysis session. Once it became clear that the participants understood the tool, they were taught constituted a design defect how to find design it in the interface specification.

Participants were first taught the *consistency* principle of interface design. Several examples of how it could be violated were demonstrated in the text editor specification used earlier. During this training, no mention was made of how the design defects could be classified as input, output and control flow defects. However, the examples chosen to illustrate design defects contained at least one of each type from the classification. An important part of the training was that participants were given a suggested analysis strategy; heuristics that would facilitate the finding of design defects. The strategies were developed through experience with the specification tools (See the strategies in the appendices describing the training materials). Again, the goal was to give the best possible training.

During the last part of the training session a short test was given to see if the participants were proficient at understanding the notation and meaning of the specification tool. Those participants that scored less than 90% correct on the exam

were eliminated from the study. No participants were eliminated by the proficiency exams. The exams did not test the participant's ability to find design defects. (See appendices F and G for the SFG and GTN proficiency exams.)

During training questions were encouraged and some free exchange between the instructor and the participants occurred. The data collection session was scheduled for the second day after completion of the training session.

Data collection session. Before beginning the one hour analysis session, participants were told in a general way what the RAMCAD interface appeared and the special conventions and words they would see in the specification. The purpose was to familiarize the participants with the main components of a windowing CAD system. Without this understanding, the specification would be very unclear.

Next the subjects were told the procedure they should follow during the analysis. Participants were required to keep track of the time when they found a particular design defect. When they thought they had discovered a defect, they were to circle it and write down the current time to the nearest second. During the rapid-prototype sessions, participants spoke aloud as they worked and indicated when they had found a defect. Times were recorded directly on the video tape using a time/date generator. Later on, the sessions were reviewed on video tape and the times transcribed. The performance measure were calculated from these data.

After a brief period to answer any last minute questions, participants were separated and they began the analysis of the interface specification. Participants in the rapid-prototype group were asked to run the prototype through the task scenario at least three times. None of the participants felt they needed more than the one hour time allotment. (See Appendices H,I, and J for the materials used by the participants during the analysis session).

Following the analysis, participants were brought together and shown the same interface specified with the other two specification tools. The training materials used for the other tools were handed out for reference. This allowed side by side comparison of the three specification tools. Because of the group setting, it was not be possible for

everyone to interact with the rapid-prototype. However, all participants observed the prototype being described.

The last part of the data collection session was the administration of a questionnaire concerning the analysis session and the specification tools (see Appendix K for the questionnaire). The first part of the questionnaire asked the participants to assess the general level of mental workload and difficulty they encountered during the analysis period. Next they were asked to rank the specification tools across input, output and control flow dimensions on the same criteria used in the engineering analysis. This was done to see how their rankings compared to the those in the tradeoff analysis. At the end of the data collection session a short debriefing was given to encourage the participants not to discuss the study among their peers that were involved in the study.

RESULTS

Performance Data

Three performance measures were calculated from the human factors analysis data: the percent of the total number of imbedded design defects found by a participant; the number of false-alarms; and the total time needed to complete the analysis. Because the distribution of imbedded design defects was not symmetric between the three specifications, a percentage measure was used to evaluate performance instead of raw scores.

During the process of transcribing the verbal and written records of the analysis sessions, it was found that decision rules had to be developed that would determine whether a participant had found an imbedded design defect or had made an error. In the case of the rapid-prototype group, participants were encouraged to use precise language, but, participants found it difficult to use anything but general terms to describe a defect. As such, a defect detection was awarded if they stated that they had discovered a design defect and that defect related in a general way to a real imbedded design defect. The SFG and GTN groups were awarded a detection if they had marked the specification either at the first appearance of a design defect or at any instantiation of that defect existing in

other parts of the specification. For all groups a false-alarms was awarded if the participant falsely reported that a particular design defect existed in a specification. If the participant later reported finding the same false defect in another part of the USI specification, they were not awarded an additional false-alarm.

Overall measures. Table 8 summarizes the collected performance data. Detection performance is broken down by the type of design defect found (input, output, or control flow). Note that there were not any imbedded output design defects in the GTN specification so there was not applicable performance data. One participant in the GTN group performed much worse that other participants. In a casual discussion after the study with the participant, it was clear that this individual had not fully comprehended that the RAMCAD interface was a multi-tasking and windowing interface. It was believed that without this understanding, analysis performance was severely degraded Later analyses revealed that significant results were not sensitive to including or excluding this participant's data.

The three performance measures were analyzed to determine if there were significant differences between the three groups. The results of the one-way analysis of variances (ANOVA) revealed significant differences in false-alarms, $F(2,9) = 5.15$, $p < 0.05$, and in total time, $F(2,9) = 6.88$, $p < 0.05$. Post hoc t test comparisons showed that for overall times, the rapid-prototype and GTN groups were significantly ($p < 0.05$) different from the SFG group. Comparisons of false-alarms showed that the SFG group made significatly less mistakes that the rapid-prototype group. Other comparisons of false-alarms were not significant. On average the SFG group took more time to analyze the specification and but less errors. Overall detection performance differences were not significant. Thus, the groups were considered to be equivalent in finding design defects, however, a further analysis was performed taking into account the type of design defect.

TABLE 8

Summary of Performance Data for Each Participant and Specification Tool

Rapid-Prototype

Participants	*Percent Defects Found**	*False-Alarms*	*Total Time (min.)*
1	63.6 [100.0, 80.0, 25.0]	13	38:50
2	54.6 [100.0, 60.0, 25.0]	9	41:50
3	63.6 [50.0, 80.0, 50.0]	3	37:10
4	63.6 [100.0, 100.0, 25.0]	6	41:40
MEANS	61.4 [87.5, 80.0, 31.3]	7.75	39:52

Semi-Formal Grammar

Participants	*Percent Defects Found**	*False-Alarms*	*Total Time (min.)*
1	63.6 [100.0, 0.0, 60.0]	0	59:10
2	81.8 [100.0, 100.0, 60.0]	0	49:00
3	81.8 [100.0, 100.0, 60.0]	2	58:00
4	54.6 [100.0, 50.0, 20.0]	1	58:00
MEANS	70.5 [100.0, 62.5, 50.0]	.75	56:02

Generalized Transition Network

Participants	*Percent Defects Found**	*False-Alarms*	*Total Time (min.)*
1	60.0 [60.0, n.a., 60.0]	3	50:50
2	70.0 [80.0, n.a., 60.0]	2	40:00
3	70.0 [60.0, n.a., 80.0]	5	51:00
4	20.0 [20.0, n.a., 20.0]	9	30:00
MEANS	55.0 [55.0, n.a., 55.0]	4.75	42:57

*Percent defects found are read as:
TOTAL [INPUT ONLY, OUTPUT ONLY, CONTROL FLOW ONLY]

Defect detection performance and the system model. A one-way ANOVA was performed on the performance data representing both the type of specification tool and type of imbedded design defect. If all combinations of tool and defect type were considered, there would be nine groups of data, however, there were not design defects in the output component of the GTN tool so this group was not included. The ANOVA on the eight groups of performance data was significant, $F(7,24) = 3.22$, $p < 0.05$. A post hoc comparison of the groups using the t test showed significance, $t(24) = 2.06$, $p < 0.05$, for several of the comparisons. Table 9 shows the matrix for the comparisons and the significant cells. Note that comparison pairs are separated by a diagonal line. The system model component below the diagonal line referes to the specification tool to the left. The comparison is the made to the system component above the diagonal and corresponds to the specification tool heading the column. Of particular interest are the significant detection performance differences between the rapid-prototype group's input (mean = 87.5) and control flow (mean = 31.3), and the output (mean = 80.0) and control flow (mean = 30.3). In other words, the rapid prototype is relatively poor at representing control flow as compared to its ability to represent input and output. This result was predicted in the engineering tradeoff analysis. The SFG group showed significant comparisons between the input (mean = 100.0) and output (mean = 62.5) detection performance, and the input (mean = 100.0) and control flow (mean = 50.0) detection performance. The SFG is better at representing input as compared to its representation of output and control flow. The rankings in the tradeoff analysis support this result. The other significant comparisons shown in Table 9 that are across specification tools do not have useful interpretations.

TABLE 9

Results of Post Hoc Comparison t-Tests on Percent Defects Found for the System Components: Input (I),Output (O), and Control Flow (C-F)*

Specification Tools	*Specification Tools*		
	Rapid-Prototype	*SRG*	*GTN*
Rapid-Prototype	C-F I	C-F O	C-F I
SRG	C-F I	O I	C-F I
GTN		C-F I	C-F I

*All comparisons shown are significant at the .05 level

Comparisons can be read by finding the system components paired together above and below the diagonal lines. A system component below the diagonal is associated with the specification tool to the left. A system component above the diagonal is associated withthe specification tool at the top of the column.

TABLE 10

Defect Detection Performance Data by Specification Tool and Design Defect Type

Rapid-Prototype

	Input					*Output*					*Control Flow*				
Participants	1	2†	3†	4	5	1	2	3	4	5	1†	2†	3†	4†	5
1		•	•			•	•		•	•	•				
2		•	•			•	•		•					•	
3			•			•	•		•	•	•			•	
4		•	•			•	•	•	•	•			•		
TOTALS	*	3	4	*	*	4	4	1	4	3	2	0	1	2	*

SFG

	Input					*Output*					*Control Flow*				
Participants	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
1	•	•	•		•						•	•	•		
2	•	•	•		•		•			•	•	•	•		
3	•	•	•		•		•			•	•	•	•		
4	•	•	•		•					•	•				
TOTALS	4	4	4	*	4	*	2	*	*	3	4	3	3	0	0

GTN

	Input					*Output*					*Control Flow*				
Participants	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
1		•		•	•						•	•	•		
2	•	•	•		•						•	•			•
3		•		•	•						•	•	•		•
4		•										•			
TOTALS	1	4	1	2	3	*	*	*	*	*	3	4	2	0	2

*A particular imbedded design defects was not present in the specification.
†This imbedded design defect is present in all three specifications.

A careful examination of the time line transcriptions of the defect detection data did not reveal any patterns, groupings, or trends that could help interpret the performance data. Participants did not systematically find particular defects more quickly than others. It did not matter if one looked within groups or considered participants across all the groups, defects were found with little regard to the order that they appeared in the specification. One would assume that an expert performing the analysis would find defects near the front of the specification first. Defect detection performance appeared almost random when compared to time. A different examination of the defect data revealed that participants systematically failed to find some design defects.

Table 10 contains the raw defect detection data for each specification tool and participant. In the table all fifteen imbedded design defects are tabulated and divided into input, output, and control flow. Notice the dagger symbol near defect numbers. This means that in this instance, the defect was present in all three specifications. For each defect, column totals are given for how many participants found the defect. An asterisk in a column total indicates that the defect was not in the specification because it could not be represented by the specification tool. One can see that there were eight cases where participants in a group failed to find a defect. A failure being defined when less than half of the participants (i.e., one or none) detected a particular design defect (Appendix B contains descriptions of each design defect). In the rapid-prototype group, participants performed poorly at finding output defect 3, and control flow defects 2 and 3. Output defect 3 was particularly tricky to find. A group of menu items that was common to many similar contexts in the interface was in an inconsistent order in one context. The two control defects related to the behavior of a menu.

In the SFG group there were two control flow defects that were not detected (control flow 4 and 5). Control flow defect 4 was not much different than another control defect found by all participants. However, control flow defect 5, manifested itself differently than the other defects. It appeared not as inconsistent wording (unlike most inconsistencies appearing in an SFG), but, simply as a missing action-effect rule in a context. Apparently it was easier to find wording differences than a missing rule.

The GTN group had trouble finding two input defects (1 and 3) and one control flow defect (4). The two input defects were not particularly unique manifestations of design defects. In both cases, they were very similar to other input defects that the participants found with less difficulty. Control flow defect 4 is the same defect that was completely missed by the SFG group.

Questionnaire Data

Task ratings. Participants were asked questions about the human factors analysis task. They were given a five-point Likert-type scale with "1" anchored by such words as "Extremely Easy" and "5" anchored by the words "Extremely Difficult". The words on the scales varied according to the category of the rating (See Appendix K for the questionnaire). There were four categories: "Difficulty" of the task, "Task Complexity", "Mental workload", and "Stress level".

A Kruskal-Wallis one-way analysis of variance by ranks across all four categories, showed significance, Chi-square approximation (2) = 7.26, $p < 0.05$. Further analyses revealed significance only for task complexity, Chi-square approximation (2) = 7.89, $p < 0.05$, and mental workload, Chi-square approximation (2) = 8.48, $p < 0.05$. These analyses show that the GTN analysis task was rated more complex than the rapid-prototype analysis task, and that the SFG was rated as requiring a higher mental workload than the rapid-prototype.

Tradeoff analysis. Participants were also asked to perform a tradeoff analysis on the three specification tools using the pragmatic criteria developed earlier for the engineering tradeoff analysis. Three tables were completed by the participants. one for each of the USI components. The ranking procedure was very much like the one used in the engineering tradeoff analysis. Table 11 summarizes the results of the tradeoff analysis. In the table, all rankings from the participants and groups have been collapsed into mean responses. Notice that the overall ranks do not correspond completely with the ranks in the engineering tradeoff analysis (shown enclosed in parentheses), in particular, the ranking of the SFG as third instead of the second. The rapid prototype tool was ranked substantially higher than the other tools. The Efficiency rating for the

rapid-prototype is high because the participants weren't involved in the development of the specifications and overated the ease of constructing a rapid-prototype. A closer look at the ranking data showed that the only time that the rapid-prototype was not ranked number one was by the rapid-prototype group itself on the control flow component. This result is supported by the performance data for the rapid-prototype group which performed significantly poorer at detecting control flow defects than at detecting other types of defects.

TABLE 11

Summary of Questionnaire Analysis of Tools Across Input, Output and Control Flow*

	Understandability	*Efficiency*	*Expressive Power*	*Fidelity*	*Totals*	*Ranks*
Rapid Prototype	13.3 (13)	11.8 (9)	12.7 (8)	13.5 (11)	51.3 (41)	1 (1)
Semi-Formal Grammar	8.6 (11)	7.9 (13)	8.4 (7)	6.3 (6)	31.2 (37)	3 (2)
GTN	9.6 (9)	9.8 (10)	8.7 (9)	7.7 (6)	35.6 (34)	2 (3)

*Means are calculated across all participants. Numbers in parentheses are the equivalent rankings from the engineering tradeoff analysis.

DISCUSSION

The specialized nature of the participant population dictated that only a small number of participants could be used in the study. With this small number of participants, it was not expected that significance would be seen in the results, but, since significance was found, it was concluded that something compelling was going on. The purpose of the study was to determine the extent to which the choice of USI specification tool affected the performance of a human factors engineer, and that the engineering tradeoff analysis would partially predict the effect. The study did not generate significant performance data to say if the rapid-prototype, SFG, or GTN is the best tool for representing output, input, or control flow,.respectively. What was learned from the study, however, is that the tools varied significantly in their ability to represent the components of the USI. The process of constructing the three RAMCAD specifications and generating design defects illuminated some of tools' limitations. The performance data further clarified each tool's strengths and weaknesses. The results of study hold implications for how specification tools should be used as part of the USI design process.

It is important to keep in mind that the study looked at the representation of only one class of design defect (albeit one recognized as having a large affect on usability), this fact calls into question the generalizeabilty of the results to the overall USI specification and design process. This author has taken the position that defects in consistency cause a large enough portion of the usability problems in an interface that regardless of the magnitude of usability problems caused by other defects in the design, consideration should be taken during the specification of the interface such that detection of consistency defects is facilitated. Further, consistency defects are very amenable to analytic detection methods that can be used early in the design process, while most other classes of design defects recognized by the human factors field, such as those involving mental workload and conceptual structure, are currently found using empirical techniques late in the design cycle after most of the specification work has been completed.

The Rapid-Prototype in USI Specification

The past five years had seen rapid-prototyping becoming a standard design feature of the software design process (Harker, 1987). Of the USI specification tools studied it is the one most associated with the human factors field. The questionnaire data supported the engineering tradeoff analysis result that overall the rapid prototype specification tool is the preferred USI specification technique for human factors engineers. The results of the study illuminate some of the weaknesses that rapid prototyping tools currently have and suggest caution before relying too heavily on a prototype for the specification of a USI.

The defect detection results within the rapid-prototype group agree with the engineering tradeoff result that the rapid prototype specification is weak at representing control flow as compared to its ability to represent both input and output. One reason is that the rapid-prototype group did not see an explicit representation of control flow in the rapid-prototype; they had to infer the control flow from the prototype's behavior as they interacted with it. The control flow was coded as a high level programming language and was not readable by the human factors engineers. (A very robust rapid prototyping tool, GIDS, described by Overmeyer and Campbell (1984), contains a state table representation of the control flow that can be read and is frequently designed by the human factors engineer. GIDS is an exception among rapid-prototyping tools. However it is logical to suppose, that as the prototype becomes more and more robust and complex, a human factors analysis of the state table would quickly become intractable.). To counter this weakness in a rapid prototype specification, one recommendation is to use one of the other two specification tools to specify control flow.

There is not statistical evidence to support the selection of either the GTN or the SFG tools, however, a logical case can be made for using a GTN instead of a SFG specification to augment and enhance the rapid prototype specification of the USI. The GTN, in form and content, is a familiar specification tool to the computer scientist and programmer; while the SFG specification is unfamiliar to them and lacks precision, leaving room for ambiguity and miscommunication about the design. In the design process it is the programmers who must translate the USI specifications into a finished product. A

specification that is clear and usable to the programmers has added value over a less usable specification. The GTN is the logical choice to augment the rapid prototype specification because the programmer is already prepared (or with minimal training) to interpret the GTN diagrams.

The Costs of USI Analysis

The large number of false-alarms reported by the rapid-prototype group is interesting from a cost tradeoff viewpoint. In a real design situation, each design defect that the human factors engineer finds must be checked and investigated. Each false-alarm adds to the time and cost of analyzing the interface. As stated in the introduction to this thesis, it is desirable to remove as many of the design defects from the specification before the interface software is written and design changes become costly. An analogous situation involves the construction of the rapid prototype. Experience shows that it can be a time consuming process to build a rapid prototype (possibly twice as long as a GTN or SFG specification) and make changes. The analysis of this prototype can generate many false-alarms. The more actual defects in the prototype, the more difficult it will be for the human factors engineer to detect the inconsistencies, causing more false-alarms to be generated. This suggests that even before constructing the rapid prototype, there should exist some detailed specification that can be analyzed. The tradeoff may be between spending additional time specifying and analyzing the USI with simpler specification tools first, or to go ahead early with the development of rapid prototypes.

Improving the Human Factors Analysis Process

Various USI specification tools are available in the design process. There are, however, problems in the process of analyzing a specification for design defects. It is evident from the many defect detection failures and false-alarm errors that there is room for improvement. The results of this study point to two likely explanations for the problems that participants had detecting defects: (1) the participants were not experienced enough to perform the type of rigorous analysis that would find most of the

design defects; and/or, (2) the specification tools were not good at representing the special characteristics of difficult to find defects. It is probably true that both of these explanations are factors that can sufficiently explain the data. The first explanation can account for many of the problems that the participants had. They were trained for only two hours and asked to perform a difficult and novel task. With more experience the participants would undoubtably improve their performance. What is needed in addition to more experience, is to provide the human factors engineer with a set of analysis techniques and support materials. The characteristics of the USI specifications tools determine what is needed to support the analysis process.

The difficulty that the rapid-prototype group had with one of the output defects could be solved with some sort of bookkeeping technique that would allow the human factors engineer to make notes about objects that should be checked for consistency across display contexts. The fact that many prototypes are scenario driven can make finding control flow defects particularly difficult to find. Those design defects that do not directly hamper the accomplishment of the next step in the scenario, require extra effort on the part of the human factors engineer to find. Unless the engineer is very rigorous, inconsistent behaviors in the prototype can remain hidden. This argument may account for the problems that participants had finding two control defects in the rapid prototype.

Refinements should be made in the heuristics used in the study for finding defects. As an example, one of the SFG control defects that was not found, might have been discovered if participants had been told to look carefully for missing action-effect rules in similar display contexts. Instead, the participants focused primarily on wording inconsistencies. A possible approach to solving this problem would be to identify the common characteristics of problematic defects and teach the human factors engineer better methods for locating them. A taxonomy could be prepared that lists the attributes of *consistency* design defects for each component of the USI and matches them with analysis strategies and particular specification tools.

One control flow defect was completely missed by the both the GTN and SFG groups. The one aspect of this defect that is different than the others is that it required a higher

level of understanding of the purpose and functionality of the RAMCAD interface. If it is difficult to understand the purpose of the context where a defect resides, it is then difficult to determine if there really is an inconsistency present or if it is just a novel portion of the interface that is substantially different such that comparisons to other contexts are not useful. A rigorous analysis of a USI specification is not possible without understanding the purpose and functionality the proposed USI. Interestingly, a large number of inconsistencies can be found with a very minimal understanding of the USI. In most cases it is just a matter of pattern matching (either comparing blocks of text or graphical objects). Armed with useful bookkeeping techniques, proven analysis strategies, and a good understanding of the USI's functionality, the human factors engineer would be well prepared for the task of finding design inconsistencies in a USI specification.

CHAPTER 5. CONCLUSIONS

As human factors engineers become more actively involved in the initial design process --as opposed to involvement mainly in evaluation and usability testing-- and called upon early for analysis and interpretation, it is increasingly important that they be able to communicate with other members of the design team, especially the programmers who have traditionally made many of the design decisions that impact usability. The human factors engineer needs to be prepared to analyze and interpret USI specifications. Experiences gained while preparing the specifications for the behavioral study, and the results of the study, suggest the need for improvements in USI specification tools and in the ways they are used by the human factors engineer.

FUTURE DIRECTIONS FOR USI SPECIFICATION TOOLS

Rapid-prototyping is quickly becoming a permanent part of the USI design process. The results of the study suggest that a rapid-prototype does not adequately represent the control flow component of the interface. To counter this weakness a Generalized Transition Network (GTN) should be used to augment the rapid-prototype. In the future, a better solution may be to integrate the GTN into the rapid-prototyping tool. With this configuration, the GTN diagram would be generated as an integral part of the rapid-prototype. The GTN would drive the rapid-prototype providing behavior for the inputs and outputs designed with other utilities in the rapid-prototyping tool. The human factors engineer could then perform an analysis of the rapid-prototype looking for design inconsistencies and having the advantage of an explicit representation of control flow imbedded into the design.

During the construction of the GTN specification, it became clear that many design inconsistencies could be found using a computer algorithm. GTNs can be represented in a

form readable by computers (Kieras and Polson, 1985) and as Foley (1987) has demonstrated, design defects in consistency can be found by the computer. If design inconsistencies can be found in computerized GTNs, then the first part of analysis of the specification would be to have the computer make a pass at the specification and flag design defects. After this, the human factors engineer would verify the work of the computer and look for design inconsistencies that the computer could not find.

The RAMCAD task was selected for its diverse inputs, outputs, and control flow. The system was primarily menu-driven, and so the syntax of the command language was very simple. Many USIs contain very complex and powerful command languages. The three specification tools used in the study are not typically used to represent the syntax of a large command language. Traditionally, the computer scientist designs the command language of a system. Computer scientists are familiar with this task and have automated tools to generate the logic programs needed to implement a new command language. BNF or BNF-like tools are used to represent the syntax of a new command language. This process is well established but not understood by the human factors engineer. As indicated in the tradeoff analysis, BNF specifications are difficult to read and are not usable by the human factors engineer. Human factors engineers need a new specification tool, or the novel use of an existing tool, so that they can analyze the syntax specification of large command languages before they are coded.

Improvements in the capabilities of specification tools must accompanied by improvements in the process of human factors analysis. Currently, the human factors engineer is not used to seeing USI design specifications of the kind used in the study. Part of a human factors engineer's education should be learning how to interpret and use USI specification tools. This thesis reviewed the major USI specification tools and through a tradeoff analysis concluded that three tools might be potentially useful to the human factors engineer. The study showed that it is possible for a human factors engineer to perform an analytic analysis of a USI specification, but, for the analysis to be successful, the human factors engineer should be prepared with: (1) good analysis strategies, (2) bookkeeping techniques, and (3) detailed knowledge of the USI's functionality.

THE USI DESIGN PROCESS AND SPECIFICATION TOOLS

The trend toward improving the usability of USIs has led to the development of new software engineering tools such as rapid-prototyping and new high-level computer programming languages. The result of this trend has been the development of comprehensive software design environments called User-Interface Management Systems (UIMS, see Olsen, 1987). A UIMS is a tool (or tool set) designed to encourage the cooperation of several specialists in the rapid development, tailoring and management of USIs. An underlying assumption of the UIMS is that the application program can be separated from its interface. This has the advantage that several interfaces can be designed for the same software application. If a USI software design environment is to be effective, it must consider the different design tool needs of the interdisciplinary design team. For example, the computer scientists may be concerned with the reusability of software code in the new design and its compatibility with existing software and hardware, while the human factors engineer (HFE) may be concerned with the compatibility of the new design with the expectations of certain user populations. Ideally any USI design environment explicitly supports the work of the HFE. This study has shown that the HFE requires design tools that have different properties than those of the computer scientist.

Any software engineering environment can be depicted as having three elements (Charette, 1986). The foundation or bottom layer is the software development *process* describes the chain of events leading to a software product. The second element is *methods.* This element includes all the methods needed to define, describe, abstract, modify, refine, and document the software. The top element of the software engineering environment is *automation.* This element refers to the use of the computer to implement the *methods*

Currently most software design environments use the traditional waterfall model of the software development process. This process is characterized by six stages: requirements analysis, preliminary design, detail design, production, testing, and maintenance. The stages have different names depending on the author, but generally have the same meanings. Each stage is considered discreet and the process does not move

on until a particular stage is completed. Modifications to the model have included incremental development stages where flexibility is allowed to build rapid-prototypes and to get early feedback from designers and users (see Williges, Williges and Elkerton, 1987).

The UIMSs being developed use a different software development process model called an Operational model. The requirement analysis stage from the waterfall model is still present in the Operation model but the design phases are different. The focus of this process is the construction of executable specifications. These specifications are successively transformed into efficient implementations (Charette, 1986). The specifications that evolve during the design process have to be understood by both the computer scientist and the HFE. This thesis has shown that the choice of specification tools is critical to the work of the HFE. How do the rapid-prototyping and GTN specification tools fit into the Operation model of software design?

Figure 7 shows how specification tools surveyed in this thesis can be used in the Operation model. In the background of the figure, the stages of the Operational model are depicted, showing the transformation of a specification into an executable software product. The foreground depicts a detailed expansion of the Operational Specification stage. The roles of both the computer scientist and the HFE are shown. They each use their own specification tools which start at a low level and evolve into tools used to create a robust specification. In some cases there are logical ways to migrate the initial specification to more to powerful specification tools. The rapid-prototype is a natural extension of a storyboard, and, a simple state diagram can be expanded into a GTN. The rapid-prototype and GTN specification tools are shared by computer scientist and the HFE. A rapid-prototype specification augmented by a GTN could be transformed automatically to the next stage in the Operational model. As the specification becomes more detailed, the level of sophistication of the analyses that can be performed on the specifications increases. The HFE can perform an analytic analysis on the storyboard augmented by a simple transition diagram and then later perform a detailed analysis on the rapid-prototype and GTN specifications like the one performed in the behavioral study.

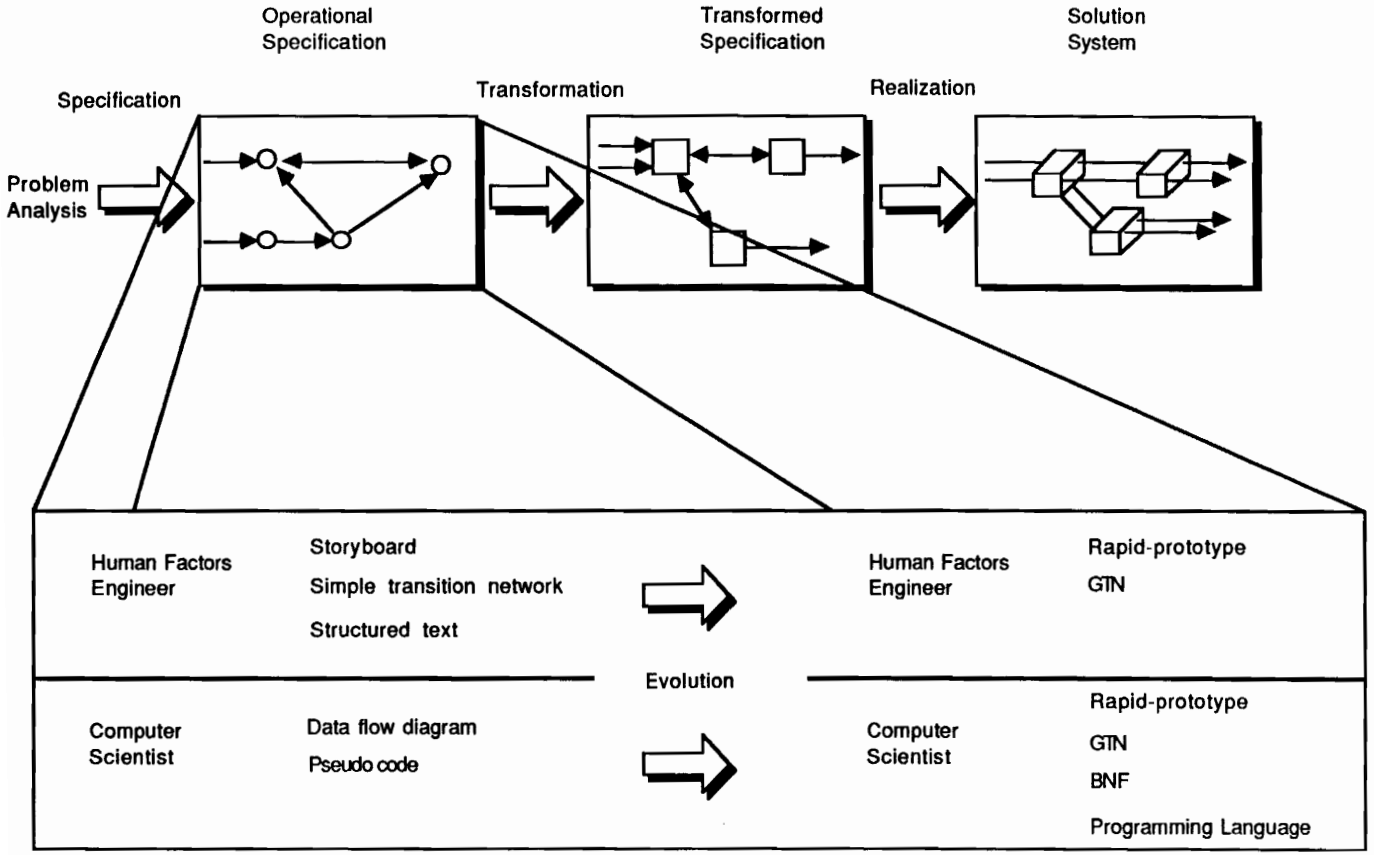


Figure 7. *The process of USI specification and an operational model of USI design (based on Charette [1986])*

CONCLUSIONS FOR HUMAN FACTORS ENGINEERING

This thesis has several conclusions that address how human factors engineers can become more effective members of software development teams, traditionally dominated by computer scientists. The engineering tradeoff analysis has shown that USI specification tools are diverse and vary in representational power and usefulness to the human factors engineer. There are many levels in the USI component taxonomy that are either, partially represented, or not represented at all by the specification tools currently available. New tools need to be developed. Without a complete specification of the USI design decision will be made on the fly and cause human factors problems that may go undetected until late in the software design process.

The behavioral study demonstrated the feasibility of an analytic human factors analysis. It is recommended that the rapid-prototyping and GTN specification tools be used together to counter limitation of these tools when used separately. The human factors engineer needs to be educated as to the use of these tools. Knowledge of these tools will improve communication about the USI design between members of the design team and make human factors analysis more systematic.

The emergence of User-Interface Management Systems is an impetus for the human factors engineer to learn USI specification tools. In these systems, specifications have the additional requirement that they be executable. As these systems are developed it is important that human factors engineers can interpret the specifications used to represent a new design. The software development environments must not become so complex as to exclude the human factors engineer from the early design stages.

References

Akmajian, A., Demers, R.A., and Harnish, R.M. (1984). *Linguistics:* an *introduction to language and communication* (pp. 390-427). Cambridge, MA: The MIT Press.

Backus, J.W. (1959). The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM conference. In *Proceedings of international conference on information processing* (pp.125-132). New York: UNESCO.

Blum, M.L., and Naylor, J.C. (1968). *Industrial psychology: Its theoretical and social foundations.* New York: Harper and Row.

Buxton, W. (1983). *Lexical and pragmatic considerations of input structures.* Computer Graphics, January, 31-37.

Card, S.K., Moran, T.P., and Newell, A. (1981). *The psychology of human-computer interaction* (pp. 139-147). Hillsdale, NJ: Lawrence Erlbaum Associates.

Carroll, J.M., and Rosson,M.B. (1985). Usability specifications as a tool in interative developement. In H.R. Hartson (ed.) *Advances in human-computer interaction* (pp. 243-281). Norwood, NJ: Ablex Publishing.

Charette, R.N. (1986). *Software engineering environments: concepts and technology.* New York, NY: Intertext Publications Inc.

Dix, A.J. and Runciman, C. (1985). Abstract models of interactive systems. In P. Johnson and S. Cook (eds.), *People and computers: designing the interface.* Cambridge University Press.

Foley, J. (1987). *Transformations on a formal specification of user-computer interfaces.* Computer Graphics, 2, 109-113.

Foley, J.D., and Van Dam A. (1982). *Fundamentals of interactive computer graphics* (pp. 218-242). Reading, MA: Addison-Wesley.

Gould, J.D., and Lewis, C. (1985). *Designing for usability: key principles and what designers think* Communications of the ACM, 3, 300-311.

Harker, S. (1987). Rapid prototyping as a tool for user centered design. In G. Salvendy (ed.), *Congnitive engineering in the design of human-computer interaction and expert systems.* Amsterdam: Elsevier.

Hartson, H.R., Roach, J.W., Callan III, J.E., Nickson, M.M., Pittman, J.A., and Yunten, T. (1986). Dialogue management as part of software engineering methodolgy. In R.W. Ehrich, and R.C. Williges (eds.), *Human-computer dialogue design* (pp. 59-85). Amsterdam: Elsevier.

Henderson, D.A. (1986). The trillium user interface design environment. In *Proceedings of CHI 1986* (pp.221-227). New York: ACM.

Jacob, R. (1983). *Using formal specifications in the design of a human-computer interface.* Communications of the ACM, 4, 259-264.

Kieras, D.E. and Polson, P.G. (1985). *An approach to the formal analysis of user complexity.* International Journal of Man-Machine Studies, 22, 365-394.

Monk, A.F. and Dix, A. (1987). *Refining early design decisions with a black-box model.* Unpublished working paper.

Moran, T.P. (1981). *The command language grammar: a representation for the user interface of interactive computer systems.* International Journal of Man-Machine Studies, 15, 3-50.

Myers, B.A., and Buxton, W. (1986). *Creating highly interactive and graphical user interfaces by demonstration.* Computer Graphics,20(4), 249-257.

Nickerson, R.S. (1986). *Using computers: human factors in information systems.* Cambridge, MA: The MIT Press.

Norman, D.A. (1986). Cognitive Engineering. In D.A. Norman, and S. Draper (eds.), *User centered system design: new perspectives on human-computer interaction* (pp.31-61). Hilsdale, NJ: Lawrence Erlbaum Associates.

Norman, D.A. and Draper, S.W. (1986). *User centered system design: new perspectives on human-computer interaction.* Hilsdale, NJ: Lawrence Erlbaum Associates.

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, 15, 379-385.

Reisner, P. (1981). *Formal grammar and human factors design of an interactive graphics system.* IEEE transactions on software engineering, 2, 229-240.

Reisner, P. (1984). Formal grammar as a tool for analyzing ease of use: some fundamental concepts. In J.C. Thoman, and M.L. Schneider (eds.), *Human factors in computer systems* (pp. 53-78). Norwood, NJ: Ablex Publishing

Roach, J.W., and Nickson, M. (1983). Formal specifications for modeling and developing human/computer interfaces. In *Proceedings of CHI 1983* (pp.35-39). New York: ACM.

Shackel,B. (1984). The concept of usability. In Bennett, J., Case, D., Sandelin, J., and Smith, M. (eds.), *Visual display terminals: usability issues and health concerns* (pp. 45-88). Engelwood Cliffs, NJ: Prentice-Hall.

Shneiderman, B. (1982). *Multiparty grammars and related features for defining interactive systems.* IEEE transactions on systems, man, and cybernetics, 2, 148-154.

Simpson, C.A., Mcauley, M.E., Roland, E.F., Ruth, J.C., and Williges, B.H. (1985). *System design for speech recognition and generation.* Human Factors, 2, 115-141.

Smith, S.L., and Mosier, J.N. (1986). *Guidelines for designing user interface software* (Report No. MTR-10090). Bedford, MA: The mitre corporation.

Wikens, C.D. (1984). Engineering psychology and human performance. Columbus, OH: Charles E. Merrill Publishing Company.

Williges, R.C., Williges, B.H., and Elkerton, J. (1987). Software interface design. In G. Salvendy (ed.) *Handbook of human factors.* (pp. 1416-1449). New York: John Wiley and Sons.

Wulf, W.A., Shaw, M. Hilfinger,P.N., and Flon, L. (1981). *Fundamental structures of computer science* (pp. 1-48). Reading, MA: Addison-Wesley.

Yunten, T., and Hartson, R.H. (1985). A supervisory methodology and notation (SUPERMAN) for human-computer system development. In H.R. Hartson (ed.) *Advances in human-computer interaction* (pp. 243-281). Norwood, NJ: Ablex Publishing.

Appendix A. Task Scenario

The RAMCAD (Reliability and Maintainability Computer-Aided Design) software is designed to combine reliability and maintainability directly with computer-aided design. The interface consists of both hardware and software components. The hardware elements include a large color graphics monitor, mouse, keyboard entry device, and a powerful engineering workstation microcomputer. The software provides an integrated environment for the designer. The system includes separate windows for the major components of the design task. Pop-up menus and a command language are provided to control the system.

A typical scenario involving the use of this interface is the design of an electronic assembly called the Gyro Roll Demodulator. In the following paragraphs a design session involving this assembly will be described to indicate how the interface might be used. This example is explained from the point-of-view of the engineer/designer using the interface.

The Gyro Roll Demodulator is a component in a large system (Stabilized Glide Slope Indicator) that includes other electronic as well as mechanical systems. The design scenario involves an electronics engineer defining a small circuit element and performing a reliability analysis on the Gyro Roll Demodulator assembly.

First, the engineer wants to get help on the main menu options. Pick "HELP" from the main menu, then "top level options" from the HELP menu. A help window appears.

After finding the information needed, the engineer is interested in the design requirements for the Stabilized Glide Slope Indicator. The engineer knows the specification window contains information about the system under development. Pick "SPEC" from the main menu, then "Current option and level" from the SPEC menu. Read the design parameters of importance.

At this point the engineer decides to complete the electronic design of the Gyro Demodulator Assembly. It is necessary to enter the CAD portion of the interface. Pick "CAD" from the main menu, then "Functional diagram" from the CAD menu.

The CAD window appears, and it contains a block diagram of the components of the Stabilized Glide Slope Indicator. Pick "ELECTRONIC ENCLOSURE" from the block diagram. A new CAD window appears containing a block diagram of the electronic elements. Pick the

"GYRO ROLL DEMODULATOR." Another CAD window appears that contains a block diagram of the circuit elements in the Gyro Roll Demodulator.

The engineer notes that one circuit element (the "FULL WAVE RECTIFIER") has not been defined. The engineer decides to call up a database of full wave rectifier circuits and to pick one to be included in the design. Pick "DATA" from the main menu then "Show circuit options" from the DATA menu.

A window appears containing schematic diagrams of full wave rectifiers. Pick one of the diagrams and then pick "ADD" from the menu in the Show circuit options window. Now that the circuit design for the Gyro Roll Demodulator is complete, the engineer wants to check the reliability of the circuit at this point. Pick "OVER" from the main menu, then pick "Current option and level" from the OVER menu. A window appears that describes overall reliability measures in both graphical and textual form.

The engineer notes that the MTBF measure meets the design criterion. The engineer wonders if this will be true if the assembly is subject to different thermal conditions. Pick "REL" from the main menu, then pick "Thermal analysis" from the REL menu.

A window appears that lists several temperature parameters used in a thermal analysis simulation. The engineer decides to change the beginning temperature. Pick the thermal parameter "25.00" in the window and type in a "10.00" as the new beginning temperature. The engineer now runs the simulation. Pick "RUN" from the menu in the Thermal analysis window.

The engineer notes that the MTBF measure in the SPEC window falls below the criterion level because of the new thermal analysis. The engineer decides to see which circuit element is responsible for this change. Pick "Reliability block diagram" from the REL menu.

A window appears containing a block diagram of the Gyro Roll Demodulator circuit elements and their reliabilities. The engineer notes that the problem is in the full wave rectifier just added. The engineer decides to save the work and leave. Pick "Enter commands" from the command line, type "SAVE", and hit the return key. Pick "EXIT" from the main menu.

Appendix B. Design Defects

Input

I1. The <esc> and <del> keys are assigned to perform the same function in different parts of the software.

I2. "QUIT" is used to close the THERMAL ANALYSIS window. "EXIT" performs the same function in other windows of the interface.

I3. The DATA top level menu, unlike all the other menus, is opened by the down button press on the mouse rather than on an up transition.

I4. Picking in the work area (main part of the window) of the SPECIFICATION window performs a window pop. All other windows that pop do so only when the title bar is picked.

I5. The <ctrl-h> keystrokes perform the backspace function in the parameter mode, while the <backspace> key performs the same function in the command input mode.

Display Output

O1. The title bar is colored green on the OVERVIEW window. All other title bars for windows are white.

O2. The menu in the SPECIFICATION window is located at the bottom of the window. All other menus are located at the top of the window.

O3. The order of the menu items in the RELIABILITY BLOCK DIAGRAM window is inconsistent with the order of the items in the other menus.

O4. The SHOW CIRCUIT OPTIONS window beeps when a circuit selection is made. No other window uses auditory feedback.

O5. The title is missing from the title bar of the HELP top level menu. All other menus have a title.

Control Flow

C1. A button lift_off on the mouse selects menu items in the DATA top level menu. Items are selected from all other menus after a touch_down + lift_off.

C2. The REL menu remains open after a selection has been made. All other menus close when any selection is made.

C3. "EXIT" on the REL menu closes not only the menu but also the OVERVIEW window. "EXIT" on all other windows closes only the menu.

C4. "Selecting "ADD" item in the CIRCUIT OPTIONS window performs the desired operation, but also closes the window. No other computing operation causes a window to close upon completion of the computing operation.

C5. The CAD windows cannot be popped. All other windows can be popped.

Appendix C. Rapid-Protoype Training

A rapid-prototype is a simulation of how a user-computer interface that is under development will look and behave. Typically, a rapid-prototype does not allow the user to use all of the functions that the final interface will contain. Instead, a small subset of the complete system is implemented, and the users of the rapid-prototype follow a scenario that allows them to perform only certain actions.

Quite often only certain input modes are supported by the prototype. For example, the real interface may allow both keyboard and mouse inputs, but the prototype may allow only mouse inputs.

GLOSSARY OF TERMS

Mouse	A pointing device that when moved by the hand simultaneously moves a pointer on the display. A mouse usually has one to three buttons on top that are used for picking.
Pick	The act of selecting an object or place on the display using the pointing device. This includes objects inside and outside any windows on the screen. The picking is performed by first moving the pointer to the desired location and pressing a button or key.
Window	A rectangular shaped region on the computer display that contains text and/or graphics. Several windows can appear on the screen at the same time. Windows can be obscured by one another. A particular instance of a window is identified by its name.
Pop a Window	An obscured window is moved on top of the windows that are covering it such that its contents can be seen completely.
Open a Window	A new window is created on the display.
Close a Window	A window is removed from the display.
Input	A string of characters entered by the user with the keyboard
Character	Any letter or symbol that can be generated using a standard typewriter style keyboard
Prompt	A symbol or character that indicates to the user where keyboard input will be displayed when entered (e.g., $, Command?, ->, etc.)
Mode	A state that the system is in that has a separate behavior from other states (e.g., the insert and replace modes in most text editors).

A PRINCIPLE OF GOOD DESIGN

During the analysis of the rapid-prototype specification you will be asked to find design defects. A design defect will be any case found in the rapid-prototype specification that violates the human factors design principle of "consistency."

consistency principle Minimize the difference in human computer dialogue within the interface. This means that similar contexts of an interface should be implemented so that the user can expect the interface to behave and appear in a uniform way.

How should one apply this principle when analyzing an rapid-prototype? The next section describes rules that one can use to find design defects in an prototype specification.

HOW TO LOOK FOR DESIGN DEFECTS USING A RAPID PROTOTYPE

Below are several techniques that you may find useful when analyzing a prototype interface for design defects.

1. First run through the complete scenario of the prototype. Learn to use the mouse to pick menu items. Get an overall idea of what the interface looks like.
2. Start from the beginning of the scenario this time looking for inconsistencies in the way objects appear on the display from one screen to the next. Remember the various attributes objects have such as color, movement, location, blinking, etc.
3. Note the way that you are required to perform actions with the system. Look for inconsistent ways of running the prototype.
4. If the prototype does unexpected things, remember where they occurred and carefully note them the next time you run through the scenario. Unexpected behavior in the prototype may indicate that there is an inconsistency in the design.

Appendix D. SFG Training

A Semi-Formal Grammar (SFG) specification is a textual representation of the user-computer interface. An SFG specification has three main components: major display contexts, an action vocabulary, and action-effect rules.

First, the interface is divided up into major display contexts. These are short description of how different modes or screens of the interface look. Second, the set of user actions is specified. This is a listing of all possible user inputs to the interface. It is in this section that the possible keyboard and mouse actions that the user is allowed to use would be defined. Finally, for each major display context, rules are defined that describe the effects that user actions have on the interface.

The following pages contain a definition of the SFG notation and an example specification of a simple text editor using the SFG.

Display Contexts

Each display context is labeled with the letter "C" (context) followed by a number that indicates which particular context it is. Similar contexts are indicated by adding a lower case letter to the label. Thus, contexts are numbered as C1a, C1b, C1c, etc.

Action Vocabulary

Keyboard inputs are surrounded in angle brackets like < and >. The item in between is a description of the key. Examples are: <space>, meaning the space bar; <A>, the letter "A"; and <backspace>, meaning the key that performs a backspace.

Sets of inputs (such as all the alphabetical characters) are represented by a name or an underlined letter (A). following the name is an equal sign. To its right in curly brackets {...} is the definition of the actions the set includes.

Action-Effect Rules

Each action-effect rule is labeled with the letter "R" (rule) followed by a number. Sub-rules are indicated by adding a lower case letter to the label. Thus, sub-rules of rule R1 would be numbered as R1a, R1b, R1c, etc. The rules are broken into an action and an effect. They are separated by two colons { :: }.
An example:

<space> :: move the cursor one character to the right.

MAJOR DISPLAY CONTEXTS FOR EXAMPLE EDITOR

C1. Welcome banner and prompt for document name.

C2. Task menu (edit, create, spelling check, print, etc.)

C3. Command mode (page with "Enter command" under a line of dashes at the bottom).

C4. Insert mode (page with no line of dashes and "insert in progress" at the bottom).

C5 to C19. Other command modes (not discussed).

ACTION VOCABULARY FOR EXAMPLE SYSTEM

A = {A,... Z, a,...z, 0,...9, !, ″, , %, &, (, ...} (Printable characters)

D = {1, 2, 3, 4, 5, 9} (Menu digits)

C = {A, a, B, b, C, c, D, d, F, f, H, h, I, i, J, j, K, k, L, l, M, m, N, n, P, p, Q, q, R, r, S, s, V, v, X, x, Z, z} (Commands, D and C are subsets of A).

<space>

<tab>

<del>

<backspace>

<retn>

arrow_keys = {<up>, <down>, <left>, <right>}

<esc>

ACTION-EFFECT RULES FOR EXAMPLE TEXT EDITOR

C1. Welcome Banner and Prompt for Document Name

R1. A :: display characters on screen to left of cursor or beginning on next line, if the edge of the screen is reached. Letters appear in uppercase.

R2. <del> :: remove character to the left of the cursor, unless none typed.

R3. <space>, <backspace>, or arrow_key :: cursor moves appropriately.

R4. <tab> :: insert 8 spaces.

R5. <esc> :: no effect.

R6. <retn> :: if only letters and numerals have been typed, change mode to C2., otherwise, beep, and display message "Error in document name", pause a short while and redraw screen as it was initially in C1.

C2: Task Menu

R7. D :: change context to new task.

R8. <del> :: no effect.

R9. anything else :: beep, display message "You did not read the available options", pause a short while, redraw screen as it was initially in C2.

C3: Command Mode

R10. D :: change mode or execute command (B,P,X,Z).

R11. <retn>, arrow_key, <backspace> :: move cursor. (Movement depends on left and right text boundaries which are the leftmost and rightmost characters in each line as it is currently positioned on the screen and do not necessarily correspond to the left and right margins or the screen boundary.)

R11a. <retn> :: move to left boundary of next line down or same line if at the bottom of the screen.

R11b. <up>,<down> :: move up or down one line unless at the top or bottom of the screen; move to the left or right boundary of the new line if the movement would have taken the cursor outside of that line's text boundary.

R11c. <left>,>backspace> :: move left if not at top left boundary of text; move to right boundary of previous line if at left boundary, unless at top boundary; hidden tabs cause larger movements.

R11d. <right> :: move right if not at bottom right boundary; move to left boundary of next line if at right boundary, unless at bottom boundary; hidden tabs cause larger movements.

R12. <del> :: no effect.

R13. <esc> :: no visible effect but next action results in beep and error message, as in R14. (This is really a rider to all the other rules).

R14. anything else :: beep, display "... is not a command", wait a short while, display "Enter Command" again.

GLOSSARY OF TERMS

Mouse	A pointing device that when moved by the hand simultaneously moves a pointer on the display. A mouse usually has one to three buttons on top that are used for picking.
Pick	The act of selecting an object or place on the display using the pointing device. This includes objects inside and outside any windows on the screen. The picking is performed by first moving the pointer to the desired location and pressing a button or key.
Window	A rectangular shaped region on the computer display that contains text and/or graphics. Several windows can appear on the screen at the same time. Windows can be obscured by one another. A particular instance of a window is identified by its name.
Pop a Window	An obscured window is moved on top of the windows that are covering it such that its contents can be seen completely.
Open a Window	A new window is created on the display.
Close a Window	A window is removed from the display.
Input	A string of characters entered by the user with the keyboard
Character	Any letter or symbol that can be generated using a standard typewriter style keyboard
Prompt	A symbol or character that indicates to the user where keyboard input will be displayed when entered (e.g., $, Command?, ->, etc.)
Mode	A state that the system is in that has a separate behavior from other states (e.g., the insert and replace modes in most text editors).

SOME NOTATION CONVENTIONS YOU SHOULD KNOW

To describe the behavior of a mouse requires some special action notation. To define a pick we need three pieces of information: where the mouse pointer is on the display; which mouse button has been pushed; and whether the button was just pushed down or just released. For example, suppose the user picks the menu item "search" that is located somewhere in the main menu of a simple text editor. The following describes the action vocabulary and the action-effect rules needed to define this occurrence.

Action Vocabulary

mouse_buttons [button, position] (describes the current state of the mouse buttons)

button = {<left>, <right>] (the two mouse buttons)

position = {up, down} (the position of a particular mouse button)

point_to_a_place [place] (The mouse pointer is moved to a particular place on the screen using the mouse)

place = {main_menu}

pick_lift_off [place] = {point_to_a_place [place] + mouse_button [<left>, down] + mouse_button [<left>, up]} (The mouse pointer is moved to a particular place and it is picked with a click)

Action-Effect Rules

R1. pick_lift_off [main_menu] :: pick a menu item from the main menu.

R1a. "search" :: search for a word and highlight it.

This example demonstrates a few conventions. First, for names that are more than one word long underscore symbols are used to join words together. Secondly, menu items are always surrounded by double quotes.

A PRINCIPLE OF GOOD DESIGN

During the analysis of the SFG specification you will be asked to find design defects. A design defect will be any case found in the SFG specification that violates the human factors design principle of "consistency."

consistency principle Minimize the difference in human computer dialogue within the interface. This means that similar contexts of an interface should be implemented so that the user can expect the interface to behave and appear in a uniform way.

How should one apply this principle when analyzing an SFG? The next section describes rules that one can use to find design defects in an SFG specification.

HOW TO LOOK FOR DESIGN DEFECTS USING AN SFG

Below are several techniques that you may find useful when analyzing a user-computer interface for design defects.

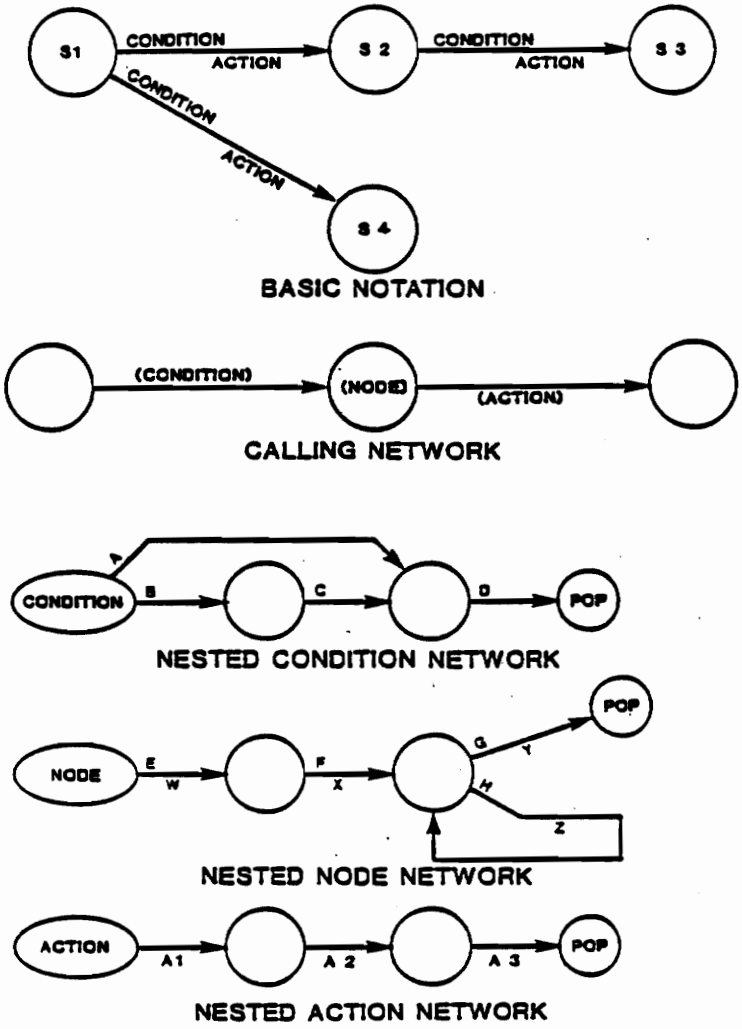
1. At the most basic level, design defects relating to inconsistency will appear as descriptions worded as exceptions. This means, for example, that if a major display context is similar to several others but not in all parts of the description, then you should take a close look and see if there is a valid inconsistency.
2. Look at the action part of the action-effect rules in similar display contexts. If the same action in similar display contexts causes a different effect, then this may mean an inconsistency is present.
3. Look at the effect part of the action effect rules in similar display contexts. If the same effect in similar display contexts is caused by a different action, then this may mean an inconsistency is present.

Appendix E. GTN Training

A Generalized Transition Network (GTN) is a graphical network representation of the user-computer interface. GTN diagrams have three main components: nodes (represented as circles), conditions (labels above the directed arcs), and actions (labels below the directed arcs).

GTN diagrams can be nested in a similar way as subroutines are in most computer programming languages. Nesting is indicated when a node or arc label is surrounded by parentheses. When a nested GTN diagram successfully reaches the special node "POP," then the sub-network is exited and the calling network is activated. This is analogous to FORTRAN when a subroutine successfully reaches the "RETURN" statement, the program returns to the calling routine.

The following pages contain a GTN description of an on-line reference manual program. It will be used as a simple introduction to the GTN notation. Information can be accessed through a TABLE OF CONTENTS (TOC) mode or an INDEX mode. Once information has been located, it is read and browsed using the READ mode. Also a keyword can be entered by the user, and the program will find any occurrence of this keyword in the manual and display the information to the user.



Examples of notation for generalized transition network diagrams. (adapted from Kieras and Polson, 1985).

GLOSSARY OF TERMS

Mouse	A pointing device that when moved by the hand simultaneously moves a pointer on the display. A mouse usually has one to three buttons on top that are used for picking.
Pick	The act of selecting an object or place on the display using the pointing device. This includes objects inside and outside any windows on the screen. The picking is performed by first moving the pointer to the desired location and pressing a button or key.
Window	A rectangular shaped region on the computer display that contains text and/or graphics. Several windows can appear on the screen at the same time. Windows can be obscured by one another. A particular instance of a window is identified by its name.
Pop a Window	An obscured window is moved on top of the windows that are covering it such that its contents can be seen completely.
Open a Window	A new window is created on the display.
Close a Window	A window is removed from the display.
Input	A string of characters entered by the user with the keyboard
Character	Any letter or symbol that can be generated using a standard typewriter style keyboard
Prompt	A symbol or character that indicates to the user where keyboard input will be displayed when entered (e.g., $, Command?, ->, etc.)
Mode	A state that the system is in that has a separate behavior from other states (e.g., the insert and replace modes in most text editors).

SOME NOTATION CONVENTIONS YOU SHOULD KNOW

To describe the behavior of a mouse requires some special GTN networks and special syntax. To define a pick we need three pieces of information: where the mouse pointer is on the display; which mouse button has been pushed; and whether the button was just pushed down or just released. For example, suppose the user picks the menu item "search" that is located somewhere in the main menu of a simple text editor. We need two GTN networks: one to describe the state of the mouse, and another to describe the mouse pointer movement and what happens when a mouse button is pushed.

This example illustrates several notation conventions:

1. Subnetworks can return and modify variables in a manner similar to programming languages. The form is
   VARIABLE NAME <- VALUE (a variables is assigned a value)
2. The condition part of an arc can test a variable to see if it equals a value of interest. The form is:
   VARIABLE NAME = VALUE (does the variable equal the value?)
3. Menu items are enclosed in double quotes.
4. Names that are more than one word long are joined together by the underscore symbol.
5. Keys or buttons are enclosed by < and >.

A PRINCIPLE OF GOOD DESIGN

During the analysis of the GTN specification you will be asked to find design defects. A design defect will be any case found in the GTN specification that violates the human factors design principle of "consistency."

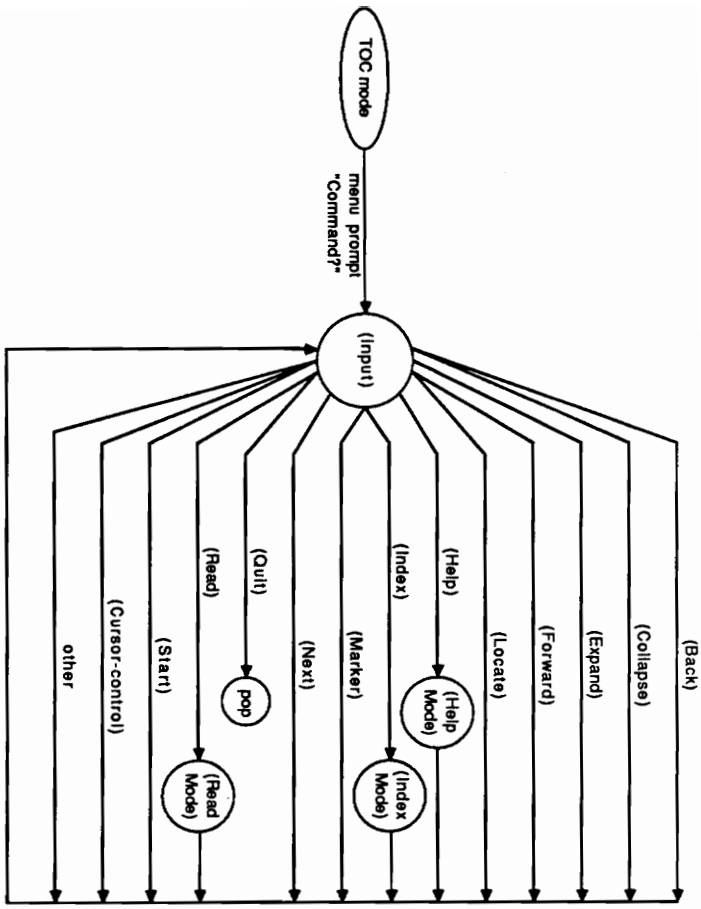
consistency principle Minimize the difference in human computer dialogue within the interface. This means that similar contexts of an interface should be implemented so that the user can expect the interface to behave and appear in a uniform way.

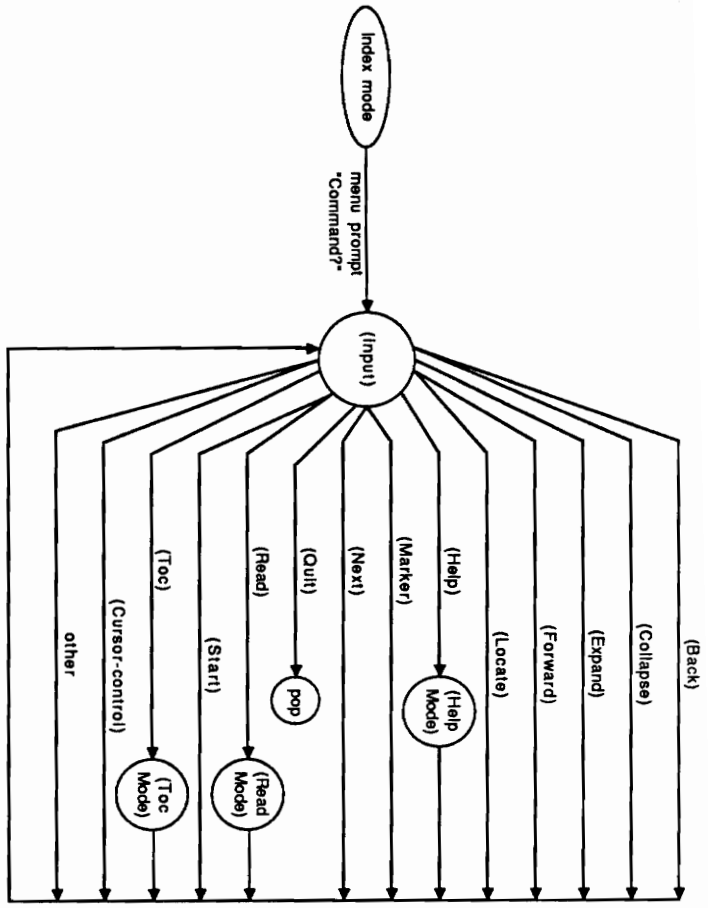
How should one apply this principle when analyzing a GTN? The next section describes rules that one can use to find design defects in an GTN specification.

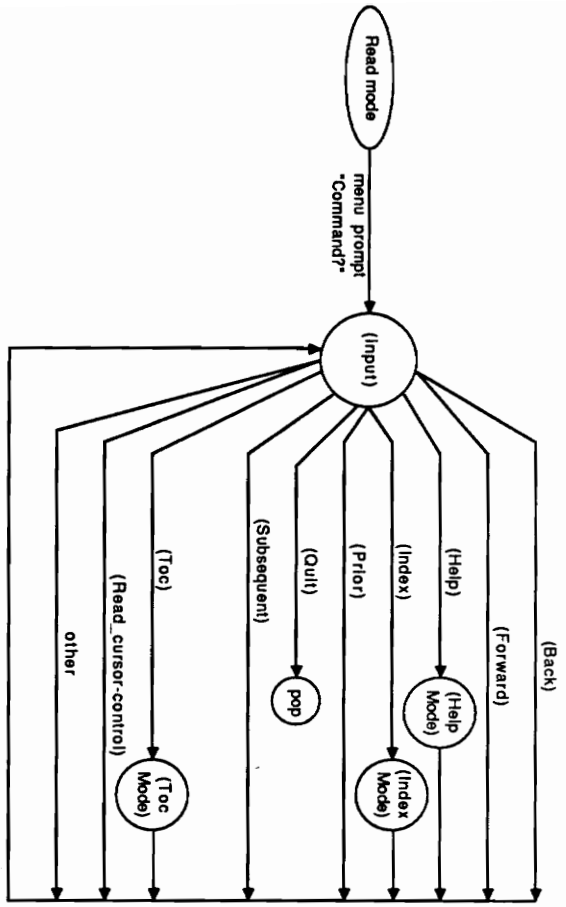
HOW TO LOOK FOR DESIGN DEFECTS USING A GTN

Below are several techniques that you may find useful when analyzing a user-computer interface for design defects.

1. Begin your analysis by looking at the upper level networks in the specification. Do not delve into subnetworks immediately. Try to get an overall understanding of the interface.
2. The GTN is primarily a graphical representation of the user-computer interface. Look for common graphical forms between networks in the specification. If you find that there is an inconsistency in the appearance of a network, then investigate it closely for inconsistencies in the interface.
3. In similar parts of the interface, the same conditions (above the arcs) should cause the same actions (below the arcs). Look carefully at similar conditions between networks and see if similar actions occur. If not, this may mean an inconsistency is present in the design.
4. In similar parts of the interface, the same actions (below the arcs) should be caused by the same conditions (above the arcs). Look carefully at similar actions between networks and see if similar conditions cause them to occur. If not, this may mean an inconsistency is present in the design.
5. In similar parts of the interface, similar flow from node to node should be present in the network diagrams. If a common node is missing from one of the networks, this could indicate an inconsistency is present in the design.


TOC mode
menu prompt
"Command?"
(Input)
other
(Cursor-control)
(Start)
(Read)
(Quit)
(Next)
(Marker)
(Index)
(Help)
(Locate)
(Forward)
(Expand)
(Collapse)
(Back)
pop
(Read Mode)
(Index Mode)
(Help Mode)


Index mode
menu prompt
"Command?"
(Input)
(Back)
(Collapse)
(Expand)
(Forward)
(Locate)
(Help)
(Help Mode)
(Marker)
(Next)
(Quit)
pop
(Read)
(Read Mode)
(Start)
(Toc)
(Toc Mode)
(Cursor-control)
other


Read mode
menu_prompt
"Command?"
(Input)
(Back)
(Forward)
(Help)
(Index)
(Prior)
(Quit)
(Subsequent)
(Toc)
(Read_cursor-control)
other
(Help Mode)
(Index Mode)
(Toc Mode)
pop

Input
Prompt
(Key_ wait)
pop

Key_wait
wait for keystroke
key <-- keystroke
pop

Back
<B>
page backward
pop

Collapse
<C>
Collpase subheading
pop

Expand
<E>
Expand heading
pop

Forward
<F>
Page forward
pop

Quit
<Q>
pop

Start
<S>
Collapse all subheadings
pop

HELP
<H>
"Press any key to continue ..."
(Key_ wait)
pop

Marker
<M>
(Prev_ Read)
pop

No previous read
Prev_read
Previous read
(Read)
pop

Cursor_control
Cursor key
Move cursor
pop

Locate
"String to locate?"
<L>
(Get_
string)
Enter
non-nul string
pattern <-- get string
(Case)
pop
Get_string
prompt
(Key_
wait)
Add char to input
Enter
pop
Case
"Consider case
(Y/N)?"
(Key_
wait)
Enter
<N>
<Y>
pop

Next
<N>
(Prev_
locate)
pop

Prior
<P>
(Prior_
Index)
pop

Previous locate
Move to old pattern
Prev_locate
No previous locate
(Locate)
pop

No prior index
Prior_Index
Prior Index
Move to prior index
pop

TOC
<T>
pop

Subsequent
<S>
(Sub_
Index)
pop

Read_cursor
_control
Cursor key
Move display one
line
pop

No subsequent reference
Sub_Index
Subsequent reference
Move to subsequent
reference
pop

Read
<R>
pop

Index
<I>
pop

Appendix F. SFG Proficiency Test

On the following pages is a Semi-Formal Grammar (SFG) specification of an on-line reference manual called the Technical Librarian. As in a reference manual there is an index. The user can browse through sections or search for specific information using the locating facility. Note that it is not a complete specification. You will be asked five true/false and five short answer questions regarding the specification. Take time to look over the display contexts. Please fill in the information below and begin the short test. You have 15 minutes. Thank you.

Name: _____

SSN: _____

Date: _____

1. In context 1, if the user strikes the "L" key, a prompt appears.

   ☐ True
   ☐ False

2. From context 1, context 4 is moved to when the "I" key is entered by the user.

   ☐ True
   ☐ False

3. In context 2, the left and right arrow keys cause lines of text to move to the left and right on the display.

   ☐ True
   ☐ False

4. There is no method to move to context 1 directly from context 3.

   ☐ True
   ☐ False

5. It is possible to exit from the Technical Librarian program from all display contexts.

   ☐ True
   ☐ False

6. In context 1, what action will cause context 4 to appear?

   _____

7. What effect does the up arrow key have in context 2?

   _____

8. What effect does the "B" key have in context 3 if the display is at the top of the page?

9. If the prompt "Consider case (Y/N)?" appears in context 1, what are the possible rule numbers that can cause this to happen?

10. In context 3, what is the action if the "F" key is pressed by the user?

MAJOR DISPLAY CONTEXTS

C1. *Table of contents mode.* Page with list of major section headings and the task menu at the bottom of the display (Back, Collapse, Expand, Forward, Locate, Help, Index, Marker, Next, Quit, Read, Start). Prompt ("Command?") just below task menu.

C2. *Index mode.* Page with hierarchical alphabetized index and the task menu at the bottom of the display (Back, Collapse, Expand, Forward, Locate, Help, Marker, Next, Quit, Read, Start, TOC). Prompt ("Command?") just below task menu.

C3. *Read mode.* Page with text of section being read and task menu at the bottom of the display (Back, Forward, Help, Index, Prior, Quit, Subsequent, TOC). Prompt ("Command?") just below task menu.

C4. *Help mode.* Page with listing of commands and their meanings for the previous mode. Task menu at bottom of the display depends on the previous mode. Prompt ("Press any key to continue...") just below task menu..

ACTION VOCABULARY

A = {A,..Z,a,...z,0,...9,!,",#,%,&,(,...}
(printable characters)

C = {B,b,C,c,E,e,F,f,L,l,H,h,M,m,N,n,P,p,Q,q,R,r,S,s,T,t}
(commands)

<space>

<tab>

<enter>

arrow_keys = {<up>,<down>,<left>,<right>}

ACTION-EFFECT RULES

C1. Table of Contents Mode

R1. B :: page table of contents backward, unless already at the top of the page.

R2. C :: collapse a heading, unless already a collapsed major section heading or a lowest level subheading.

R3. E :: expand a heading, unless no subsection under heading.

R4. F :: page table of contents forward, unless already at the bottom of the page.

R5. L, A, <enter>,Y,N :: locate a specified string in the TOC.

R5a. L :: display prompt "String to locate?" at bottom of the display.

R5b. A
:: display characters to left of the cursor at the prompt.

R5c. <enter> :: display prompt "Consider case (Y/N)?" at bottom of. the display.

R5d. Y,N :: move display to specified string, unless it does not exist.

R6. H :: move to C4.

R7. I :: move to C2.

R8. M :: move the previous page read in C3, unless no previous page read.

R9. N :: locate the next occurrence of the string specified with the Locate command (L), unless there is no next occurrence.

R10. N, A, <enter>, Y, N :: locate a specified string if a string was not previously specified with the Locate command.

R10a. L :: display prompt "String to locate?" at bottom of the display.

R10b. A :: display characters to left of the cursor at the prompt.

R10c. <enter> :: display prompt "Consider case (Y/N)?" at bottom of the display.

R10d. Y,N :: move display to specified string, unless it does not exist.

R11. A :: exit from the technical librarian.

R12. R :: move to C3.

R13. S :: collapse all subheadings in the TOC to initial state.

R14. arrow_keys :: move cursor.

R14a. <up>,<down> :: move up or down one line, unless at top or bottom of display.

R14b. <left>,<right> :: no effect.

R15. anything else :: no effect.

C2. Index Mode

R1. B :: page index backward, unless already at the top of the page.

R2. C :: collapse a heading, unless already a collapsed major section heading or a lowest level subheading.

R3. E :: expand a heading, unless no subsection under heading.

R4. F :: page index forward, unless already at the bottom of the page.

R5. L, A, <enter>,Y,N :: locate a specified string in the index.

R5a. L :: display prompt "String to locate?" at bottom of the display.

R5b. A :: display characters to left of the cursor at the prompt.

R5c. <enter> :: display prompt "Consider case (Y/N)?" at bottom of the display.

R5d. Y,N :: move display to specified string, unless it does not exist.

R6. H :: move to C4.

R7. M :: move to the previous page read in C3, unless no previous page read.

R8. N :: locate the next occurrence of the string specified with the Locate command (L), unless there is no next occurrence.

R9. N, A, <enter>,Y,N :: locate a specified string if a string was not previously specified with the Locate command (L).

R9a. L :: display prompt "String to locate?" at bottom of the display.

R9b. A :: display characters to the left of the cursor at the prompt.

R9c. <enter> :: display prompt "Consider case (Y/N)?" at bottom of the display.

R9d. Y,N :: move display to specified string, unless it does not exist.

R10. Q :: exit from the technical librarian.

R11. R :: move to C3.

R12 S :: collapse all subheadings in the index to initial state.

R13. T :: move to C1.

R14. arrow_keys :: move cursor.

R14a. <up>,<down> :: move up or down one line, unless at top or bottom of display.

R14b. <left>,<right> :: no effect.

R15. anything else :: no effect.

C3. Read Mode

R1. B :: page text backward, unless already at the top of the page.

R2. R :: page text forward, unless already at the bottom of the page.

R3. H :: move to C4.

R4. I :: move to C2.

R5. P :: move display to prior reference accessed through C2, Index mode, unless no prior references.

R6. Q :: exit from the technical librarian.

R7. S :: move display to next reference accessed through C2, Index mode, unless no subsequent references.

R8. T :: move to C1.

R9. arrow_keys :: move display text.

R9a. <up>, <down> :: move text up or down one line, unless at top or bottom of document.

R9b. <left>, <right> :: no effect.

R10. anything else :: no effect.

C4. Help Mode

R1. anything :: move to previous display context.

Appendix G. GTN Proficiency Test

On the following pages is a Generalized Transition Network (GTN) specification of a simple text editor. You can enter text, move blocks of text, find a word or group of words in the text, and delete blocks of text. Note that it is not a complete specification of a text editor. You will be asked five true/false and five short answer questions regarding the specification. Take time to look over the diagrams. Please fill in the information below and begin the short test You have 15 minutes. Thank you.

Name: _____

SSN: _____

Date: _____

1. There is only one way to enter the EDIT mode from TYPING_TASKS.

   ☐ True

   ☐ False

2. If the editor is in the EDIT mode when the user strikes the "move" key, the prompt "FIND WHAT?" appears.

   ☐ True

   ☐ False

3. In the EDIT mode, if the END condition is met, the editor saves the document and exits from the EDIT mode and returns to TYPING_TASKS.

   ☐ True

   ☐ False

4. The purpose of the INPUT mode is to get keyboard input from the user.

   ☐ True

   ☐ False

5. In the EDIT mode, if a cursor key is struck by the user, the cursor will move.

   ☐ True

   ☐ False

6. What condition must be satisfied to invoke the FIND function for the EDIT mode?

   _____

7. In the SELECT_TARGET function what action is taken when a cursor key is struck by the user?

   _____

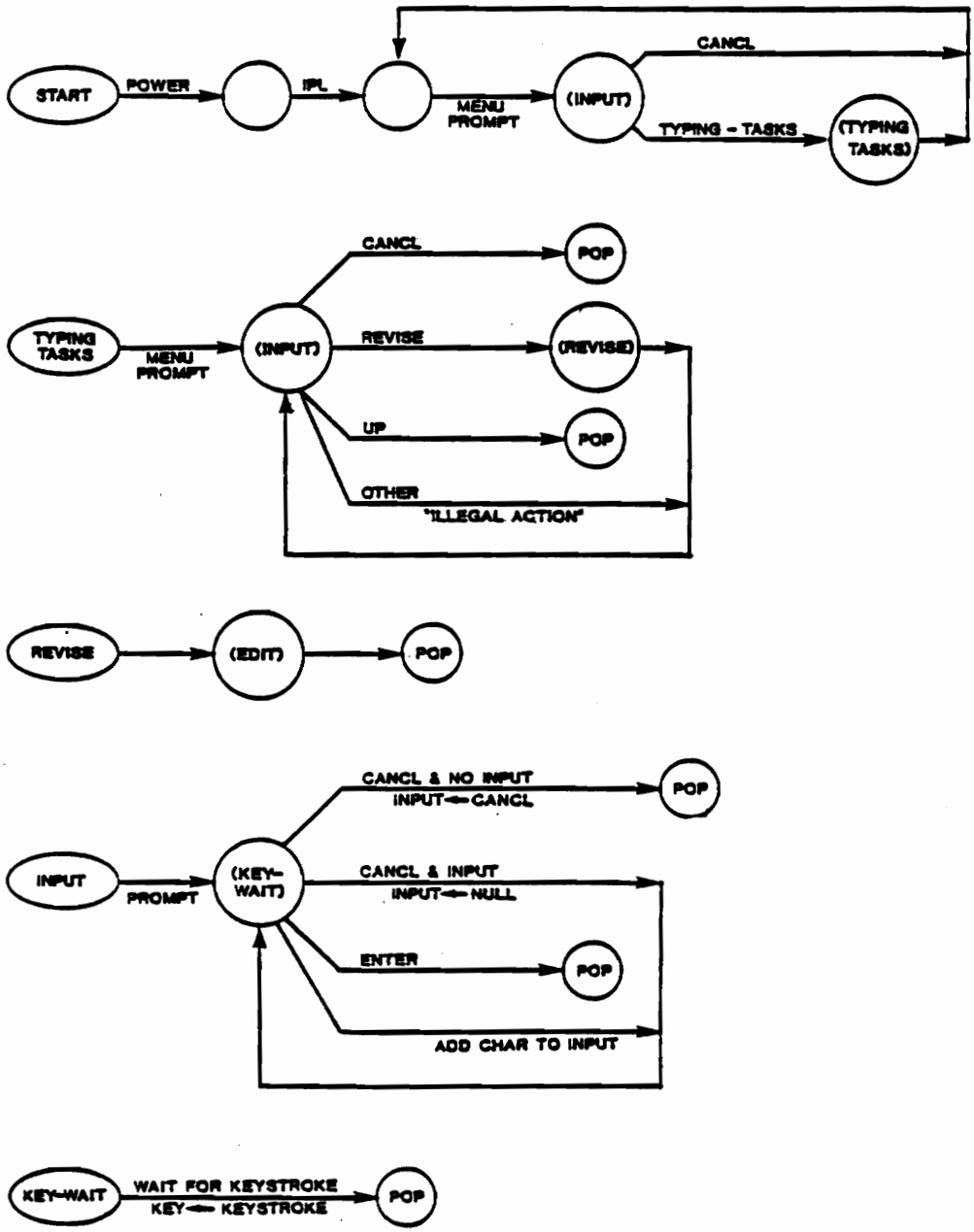
8. What is the name of the function that is responsible for getting a single keystroke from the user?

---

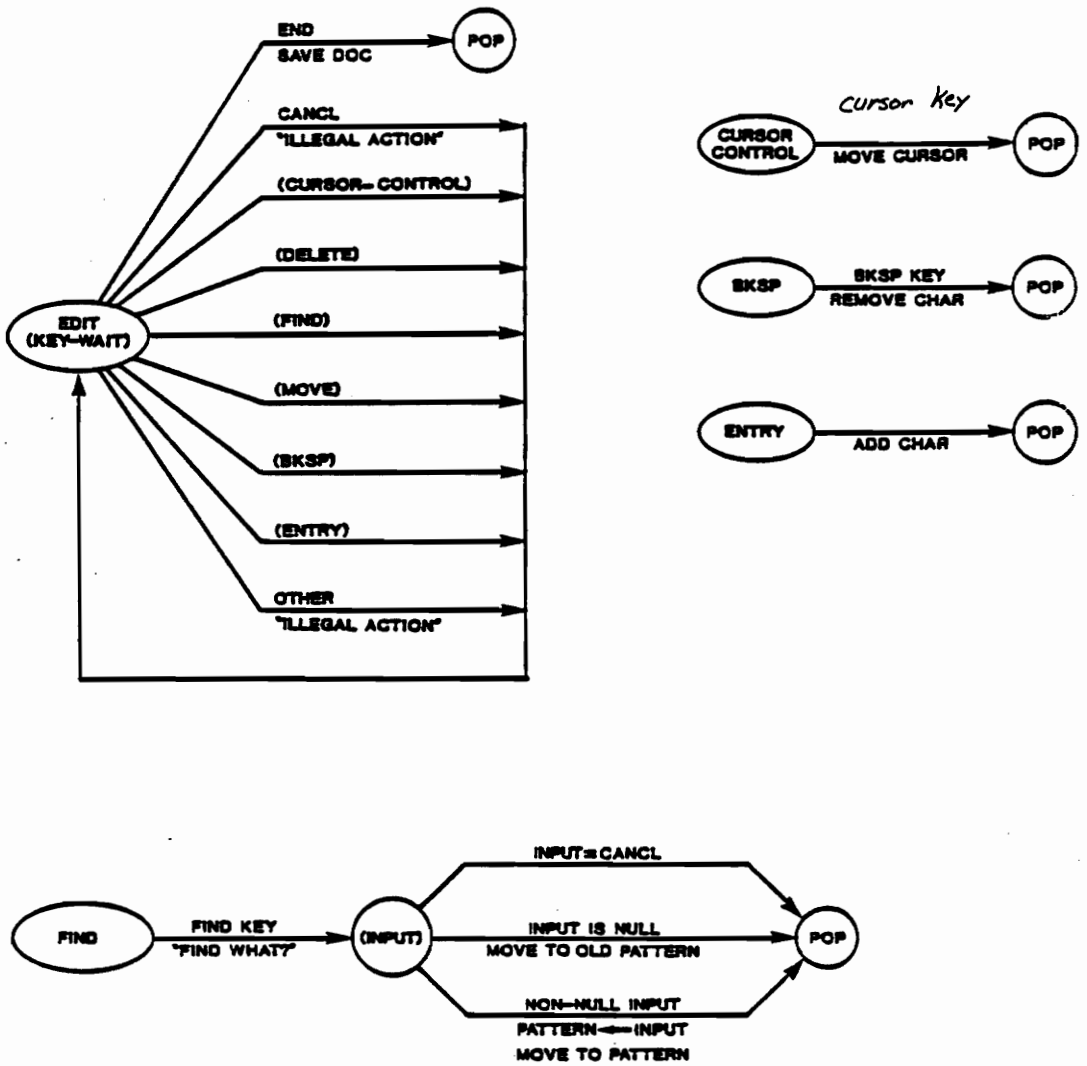
9. In the MOVE function, what is the prompt if a NON-NULL target is entered by the user?

---

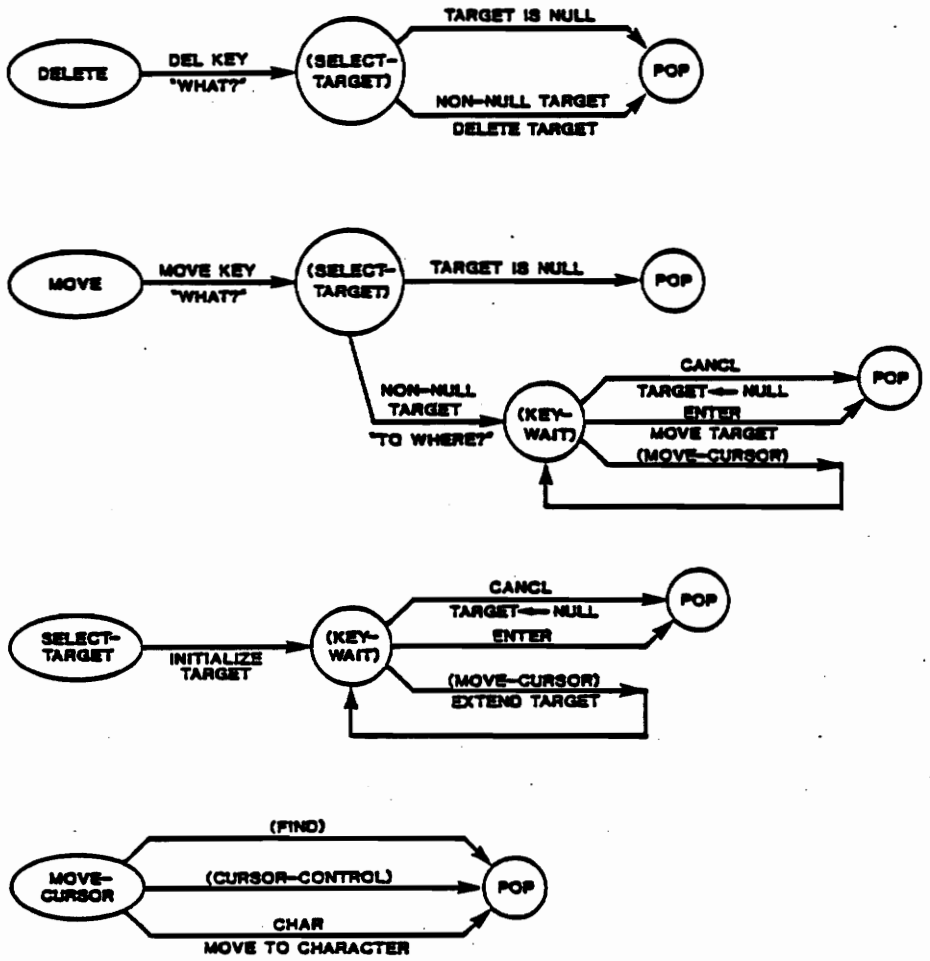
10. In the INPUT mode, what condition must be satisfied to return keyboard input to a function that called it?



Simplified GTN for the Displaywriter.
The network should be read beginning with the node labelled "START" at the top (adapted from Kieras and Polson, 1985).



Simplified GTN for the Displaywriter.
The top level of the editor is the node labelled "EDIT".
(adapted from Kieras and Polson, 1985).



Simplified GTN for the Displaywriter DELETE and MOVE functions. (adapted from Kieras and Polson, 1985).

Appendix H. Rapid-Prototype Scenario

The RAMCAD (Reliability and Maintainability Computer-Aided Design) prototype explores possible development directions to incorporate reliability and maintainability directly into the computer-aided design process. The prototype consists of both hardware and software components. The hardware elements include a large color graphics monitor, mouse, keyboard entry device, and an IBM PC AT microcomputer.

The scenario chosen for the prototype involves the design of an electronic assembly called the Gyro Roll Demodulator. The Gyro Roll Demodulator is a component in a large system (Stabilized Glide Slope Indicator) that includes other electronic as well as mechanical systems. The design scenario involves an electronics engineer defining a small circuit element and performing a reliability analysis on the Gyro Roll Demodulator assembly.

During the execution of the prototype, you will be required to pick menu items using the mouse as directed by the instructions that follow the arrow symbol "-->". The items to be picked are surrounded by < item >. Short paragraphs will be used to describe the design scenario from the user's (design engineer's) point-of-view.

The prototype implements a windowing interface (Macintosh-like).

First, the user wants to get help on the top level menu options.

--> Pick <HELP> from the main menu.

--> Pick <Top level options>.

--> Pick <EXIT> from the "HELP" window.

The user is interested in the design requirements for the entire system being designed. The specifications window contains information about the system under development.

--> Pick <SPEC> from the main menu.

--> Pick <Current option and level>.

--> Pick <EXIT> from the "SPECIFICATION" window.

The user wants to complete the design of the Gyro Demodulator Assembly. It is necessary to enter the CAD portion of the program.

--> Pick <CAD> from the main menu.

--> Pick <Functional diagram>.

The design is represented as a hierarchy of components. The user is interested in the "ELECTRONICS ENCLOSURE" which contains the Gyro Roll Demodulator. The yellow color of the box surrounding the "ELECTRONICS ENCLOSURE" indicates to the user that the design of the electronics is still under development. A green colored box means that the design has been completed. A red colored box around a functional component means that no part of the design has yet been defined.

--> Pick <ELECTRONICS ENCLOSURE> in the block diagram.

--> Pick <GYRO ROLL DEMODULATOR> in the block diagram.

An examination of the block diagram for the demodulator card shows the "FULL WAVE RECTIFIER" circuit coded in red, indicating it has yet to be specified. The user decides to call up a database of full wave rectifier circuits and selects one.

--> Pick <FULL WAVE RECTIFIER> in the block diagram.

--> Pick <DATA> from the main menu.

--> Pick <Show circuit options>.

Circuit options are listed in a column on the left side of the window. The schematic for the selected circuit is displayed on the right. At this point the user browses through the circuits until the correct one is found. This circuit is then added to the Gyro Modulator Card assembly.

--> Pick <Circuit 2> from the "OPTIONS" column.

--> Pick <Circuit 3> from the "OPTIONS" column.

--> Pick <Circuit 1> from the "OPTIONS" column.

--> Pick <Circuit 2> from the "OPTIONS" column.

--> Pick <ADD> from the "CAD/OPTIONS" window menu.

Now that the circuit design for the Gyro Demodulator Card is complete, the user wants to see how the reliability measures of the current design compare to the specified reliability measures.

--> Pick <OVER> from the main menu.

--> Pick <Current option and level>.

The user looks at the graph and sees that the MTBF (Mean Time Between Failure) reliability meets design standards for the Gyro Demodulator Card assembly. The green bar in the graph across from the MTBF measures shows that it is slightly above criterion level X. The current reliability measures are based on the thermal environment assumed for the electronic components. The user wants to change these assumptions and see their effect on reliability.

--> Pick <REL> from the main menu.

--> Pick <Thermal analysis>.

--> Pick "25.00" from the "Beginning temperatures:" line in the "THERMAL ANALYSIS" window.

The user sets the beginning temperature to 10.00 degrees at the prompt. Now the user wants to run the thermal analysis simulation.

--> Pick <RUN> from the "THERMAL ANALYSIS" window menu.

The overview window is updated to reflect the effect on reliability and other measures. The user notes the sharp fall in the MTBF measure to below standard. The user wants to see if a certain circuit component is responsible for this change. First the user wants to remove some windows from the cluttered display.

--> Pick <EXIT> from the "OVERVIEW" menu.

--> Pick <EXIT> from the "RELIABILITY" menu.

--> Pick <REL> from the main menu.

--> Pick <Reliability block diagram>.

The problem appears to be the full wave rectifier circuit added earlier. The user decides to save the work and quit.

--> Pick <ENTER COMMANDS> in the command line area.

--> Pick <EXIT> from the main menu.

The scenario is now finished. Ask the experimenter to restart the prototype for you so that you can perform an additional detailed analysis of the prototype. Remember to start from the beginning of the scenario.

Appendix I. SFG Specification

INDEX FOR SFG

MAJOR DISPLAY CONTEXTS

C1. *Main screen.* This display remains basically static, and successive display contexts are added to this display. At the top are main menu items (HELP, SPEC, CAD, etc.). At the bottom of the display is a command input area. The major portion of the display is a work area where windows appear and the user creates the device design.

C2. *Command mode.* Commands are entered from the keyboard in this context. The command line is at the bottom of the display.

C3 - C8. *Pop up menus for the main menu items.* These are small windows that contain menu items. Each window has a title bar on top (except for C3). C3, HELP menu; C4, SPEC menu; C5, CAD menu; C6, OVER menu; C7, REL menu; and C8, DATA menu).

C9. *HELP/Top level options.* This is a window with a title bar and a command menu at the top. The window contains help information.

C10. *SPEC/Current option and level.* This is a window with a title bar at the top and a command menu at the bottom. The window contains design requirement information for the device being designed.

C11a - C11*. *CAD/Functional diagram.* These are windows that have a title bar and a command menu at the top. Each window contains a functional block diagram that represents a portion of the device being designed. The amount of detail revealed about the design increases as each new CAD window is opened (e.g., C11a, then C11b, then C11c, etc.).

C12. *OVER/Current level and option.* This is a window with a title bar and a command menu at the top. The window contains summary information of the design requirements and whether the current device design meets or does not meet the design requirements.

C13. *REL/Reliability block diagram.* This is a window with a title bar and a command menu at the top. The window contains a block diagram of the functional components of the device. For each functional component reliability measures are listed for the current design level.

C14. *REL/Thermal analysis.* This is a window with a title bar and a command menu at the top. The window contains information about parameters of the thermal environment in which the device will operate. These parameters can be modified by the designer, and reliability measures will be recalculated.

C15. *DATA/Show circuit options.* This is a window with a title bar and a command menu at the top. The window contains different circuit design options that the designer can browse through and select to be included in the device being designed.

C16. *Parameter mode.* In this mode the user enters design parameters using the keyboard.

C17 - C30. Not defined for this task scenario.

ACTION VOCABULARY

A = { A,...Z, a,...z, 0,...9, !, ", %, $, #, @, (...}
(printable characters)

C = {print, directory, search, help}
(Set of commands that are made up of the set "A")

Special Keyboard Keys

<shift>
<space>
<tab>
<del>
<ins>
<enter>
<backspace>
<esc>

(arrow_keys = {<up>, <down>, <left>, <right>}

<ctrl> (control key)
<alt> (alternate key)

Mouse

mouse_buttons [button, position]
(describes the current state of the mouse buttons)

button = {<left>, <right>}
(the two mouse buttons)

position = {up, down}
(position of a particular mouse button)

point_to_a_place [place]
(cursor moved to a particular place on the screen using the mouse)

place = {main_menu, command_line, title_bar, help_menu,
help_window_menu, cad_menu, ..}
(locations on the screen that can be picked with the mouse)

pick_lift_off [place] = {point_to_a_place [place] +
mouse_button [<left>, down] + mouse_button [<left>, up]}
(cursor is moved to a particular place, and an item is picked with a click)

pick_touch_down [place1, place2] = {point_to_a_place [place1] +
mouse_button [<left>, down] + point_to_a_place [place2] +

mouse_button [<left>, up]
(cursor is moved to a particular place, and an item picked; cursor is
then moved to another place where the mouse button is released)

place1, place2 = place
(see above definition of place)

ACTION-EFFECT RULES FOR THE MAJOR DISPLAY CONTEXTS

C1. Main Screen

R1. pick_lift_off [main_menu] :: select a main menu item.

R1a. "HELP" :: add context C3 to the display. If the context already exists, then the display is not changed.

R1b. "SPEC" :: add context C4 to the display. If the context already exists, then the display is not changed.

R1c. "CAD" :: add context C5 to the display. If the context already exists, then the display is not changed.

R1d. "OVER" :: add context C6 to the display. If the context already exists, then the display is not changed.

R1e. "REL" :: add context C7 to the display. If the context already exists, then the display is not changed.

R1f. "DATA" :: add context C8 to the display, but remove quickly when the left mouse button is up.

R1g. "EXIT" :: quit program and return to the operating system.

R1h. any place else :: no effect.

R2. pick_lift_off [command_line] :: add context C2. If the context already exists, then the display is not changed.

R3. pick_touch_down [place1 = main_menu] :: select a main menu item.

R3a. "DATA" :: add context C8 to the display.

R3b. any other item :: no effect.

R3c. any place else :: no effect.

R4. pick_lift_off [work_area] :: if pick is inside a window, move to the corresponding display context, otherwise there is no effect.

R5. anything else :: no noticeable effect.

C2. Command Mode

R1.

A :: display character to the left of the cursor on the command line.

R2. <backspace>, arrow_key :: move the cursor.

R2a. <backspace> :: delete character to the left of the cursor and move the cursor left until the cursor reaches the left boundary of the command line.

R2b. <left> :: move left if not at left boundary of command line.

R2c. <right> :: move right if not at right boundary of command line.

R2d. <up>, <down> :: no noticeable effect.

R3. <esc> :: erase all characters on the command line (if there are any characters). If no characters, return to context C1.

R4.

C + <enter> :: execute the specified command. If the command is incorrect, pause, beep, and display an error message.

R5. anything else :: beep and display error message.

C3. Help Menu

R1. pick_lift_off [HELP_menu] :: select a HELP menu item, then close the menu window.

R1a. "Top level options" :: add context C9 to context C1. If the context already exists, then the display is not changed.

R1b. "EXIT" :: close the menu window and return to context C1.

R1c. any other menu item :: NOT DEFINED FOR THE SCENARIO

R1d. Title_bar :: pop the HELP menu on top if it is below and obscured by other windows.

R1e. any place else :: no effect.

R2. anything else :: no noticeable effect.

C4. SPEC Menu

R1. pick_lift_off [SPEC_menu] :: select a SPEC menu item, then close the menu window.

R1a. "Current option and level" :: add context C10 to context C1. If the context already exists, then the display is not changed.

R1b. "EXIT" :: close the menu window and return to context C1.

R1c. any other menu item :: NOT DEFINED FOR THE SCENARIO

R1d. title_bar :: pop the SPEC menu on top if it is below and obscured by other windows.

R1e. any place else :: no effect.

R2. anything else :: no noticeable effect.

C5. CAD Menu

R1. pick_lift_off [CAD_menu] :: select a CAD menu item, then close the menu window.

R1a. "Functional diagram" :: add context C11a to context C1. If the context already exists, then the display is not changed.

R1b. "EXIT" :: close the menu window and return to context C1.

R1c. any other menu item :: NOT DEFINED FOR THE SCENARIO

R1d. title_bar :: pop the CAD menu on top if it is below and obscured by other windows.

R1e. any place else :: no effect.

R2. anything else :: no noticeable effect.

C6. OVER Menu

R1. pick_lift_off [OVER_menu] :: select an OVER menu item, then close the menu window.

R1a. "Current level and option" :: add context C12 to context C1. If the context already exists, then the display is not changed.

R1b. "EXIT" :: close the menu window and return to context C1.

R1c. any other menu item :: NOT DEFINED FOR THE SCENARIO

R1d. title_bar :: pop the OVER menu on top if it is below and obscured by other windows.

R1e. any place else :: no effect.

R2. anything else :: no noticeable effect.

C7. REL Menu

R1. pick_lift_off [REL_menu] :: select a REL menu item and leave the menu window open.

R1a. "Reliability block diagram" :: add context C13 to context C1. If the context already exists, then the display is not changed.

R1b. "Thermal analysis" :: add context C14 to context C1. If the context already exists, then the display is not changed.

R1c. "EXIT" :: close the menu window and close context C14, then return to context C1.

R1d. any other menu item :: NOT DEFINED FOR THE SCENARIO

R1e. title_bar :: pop the REL menu on top if it is below and obscured by other windows.

R1f. any place else :: no effect.

R2. anything else :: no noticeable effect.

C8. DATA Menu

R1. pick_touch_down [place2 = DATA_menu] :: select a DATA menu item, then close the menu window.

R1a. "Show circuit options" :: add context C15 to context C1. If the context already exists, then the display is not changed.

R1b. "EXIT" :: close the menu window and close context C14, then return to context C1.

R1c. any other menu item :: NOT DEFINED FOR THE SCENARIO

R1d. any place else :: close the menu window.

R2. anything else :: no noticeable effect.

C9. HELP Menu

R1. pick_lift_off [HELP_window_work_area] :: no effect.

R2. pick_lift_off [HELP_window_menu_area] :: select a menu item.

R2a. "EXIT" :: close the window and return to context C1.

R2b. any other menu item :: NOT DEFINED FOR THE SCENARIO

R2c. any place else :: no effect.

R3. pick_lift_off [HELP_window_title_bar] :: pop the HELP window on top if it is below and obscured by other windows.

R4. anything else :: no noticeable effect.

C10. SPEC/Current option and level

R1. pick_lift_off [SPEC_window_title_bar] :: pop the SPEC window on top if it is below and obscured by other windows.

R2. pick_lift_off [SPEC_window_menu_area] :: select a menu item.

R2a. "EXIT" :: close the window and return to context C1.

R2b. any other menu item :: NOT DEFINED FOR THE SCENARIO

R2c. any place else :: no effect.

R3. anything else :: no noticeable effect.

C11a-C11*. CAD/Functional diagram

R1. pick_lift_off [CAD_window_work_area] :: open the CAD window that represents the portion of the functional diagram selected.

R1a. functional_block :: add next CAD window in the hierarchy to context C1 (i.e., go to next C11 context.) If the context already exists, then the display is not changed. If the functional_block is at the lowest level of the hierarchy, then block is highlighted and blinks, and no new CAD window is added to context C1.

R1b. any place else :: no effect.

R2. pick_lift_off [CAD_window_menu_area] :: select a menu item.

R2a. "EXIT" :: close the window and return to context C1.

R2b. any other menu item :: NOT DEFINED FOR THE SCENARIO

R2c. any place else :: no effect.

R3. anything else :: no noticeable effect.

C12. OVER/Current level and option

R1. pick_lift_off [OVER_window_work_area] :: no effect.

R2. pick_lift_off [OVER_window_menu_area] :: select a menu item.

R2a. "EXIT" :: close the window and return to context C1.

R2b. any other menu item :: NOT DEFINED FOR THE SCENARIO

R2c. any place else :: no effect.

R3. pick_lift_off [OVER_window_title_bar] :: pop the OVER window on top if it is below and obscured by other windows.

R4. anything else :: no noticeable effect.

C13. REL/Reliability block diagram

R1. pick_lift_off [REL_window_work_area] :: no effect.

R2. pick_lift_off [REL_window_menu_area] :: select a menu item.

R2a. "EXIT" :: close the window and return to context C1.

R2b. any other menu item :: NOT DEFINED FOR THE SCENARIO

R2c. any place else :: no effect.

R3. pick_lift_off [REL_window_title_bar] :: pop the REL window on top if it is below and obscured by other windows.

R4. anything else :: no noticeable effect.

C14. REL/Thermal analysis

R1. pick_lift_off [REL_window_work_area] :: select a thermal design parameter to be changed.

R1a. thermal_parameter :: change to context C16. If the context already exists, then display is unchanged.

R1b. any place else :: no effect.

R2. pick_lift_off [REL_window_menu_area] :: select a menu item.

R2a. "RUN" :: pause, change context to C12 and display at the top window.

R2b. "QUIT" :: close the window and return to context C1.

R2c. any other menu item :: NOT DEFINED FOR THE SCENARIO

R2d. any place else :: no effect.

R3. pick_lift_off [REL_window_title_bar] :: pop the REL window on top if it is below and obscured by other windows.

R4. anything else :: no noticeable effect.

C15. DATA/Show circuit options

R1. pick_lift_off [DATA_window_work_area] :: display the circuit schematic for each selected circuit option.

R1a. a circuit option :: beep, highlight selected circuit option, pause, then change the schematic to reflect the selected circuit option.

R1b. any place else :: no effect.

R2. pick_lift_off [DATA_window_menu_area] :: select a menu item.

R2a. "ADD" :: add circuit to the functional component selected in context C11*, then close the window and return to context C1. If a functional component has not been selected previously, pause, beep, and display an error message.

R2b. "EXIT" :: close the window and return to context C1.

R2c. any other menu item :: NOT DEFINED FOR THE SCENARIO

R2d. any place else :: no effect.

R3. pick_lift_off [DATA_window_title_bar] :: pop the DATA window on top if it is below and obscured by other windows.

R4. anything else :: no noticeable effect.

C16. Parameter mode

R1.

A :: display character to the left of the flashing cursor on the parameter line.

R2. <ctrl_h>, arrow_key :: move the cursor.

R2a. <ctrl_h> delete character to the left of the cursor and move cursor left until the cursor reaches the left boundary of the parameter line.

R2b. <left> :: move left if not at left boundary of parameter line.

R2c. <right> :: move right if not at right boundary of parameter line.

R2d. <up>, <down> :: no noticeable effect.

R3. <del> :: erase all characters on the parameter line (if there are any characters). If no characters return to context C1.

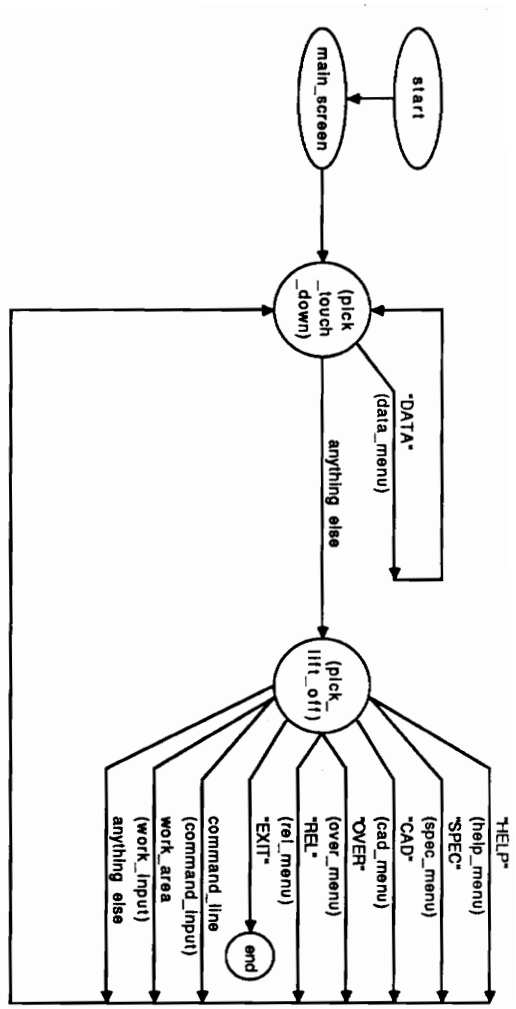
R4.

C+ <enter> :: insert the specified parameter. If the parameter is incorrect, pause, beep, and display an error message.

R5. anything else :: beep and display error message.

Appendix J. GTN Specification

INDEX FOR GTN


start
main_screen
(pick_touch_down)
"DATA"
(data_menu)
anything else
(pick_lift_off)
"HELP"
(help_menu)
"SPEC"
(spec_menu)
"CAD"
(cad_menu)
"OVER"
(over_menu)
"REL"
(rel_menu)
"EXIT"
end
command_line
(command_input)
work_area
anything else
(work_input)

command_input
prompt
(key_wait)
<esc> & no input
input <- cancel
pop
<esc> & input
input <- nul
<enter>
pop
<back space>
remove character
(cursor-control)
printable charater
add character to input
anything else
"illegal action"

parameter_mode
prompt
(key_wait)
<del> & no input
input <- cancel
pop
<del> & input
input <- nul
<enter>
pop
<ctrl-h>
remove character
(cursor-control)
printable charater
add character to input
anything else
"illegal action"

key_wait
wait for keystroke
key <- keystroke
pop

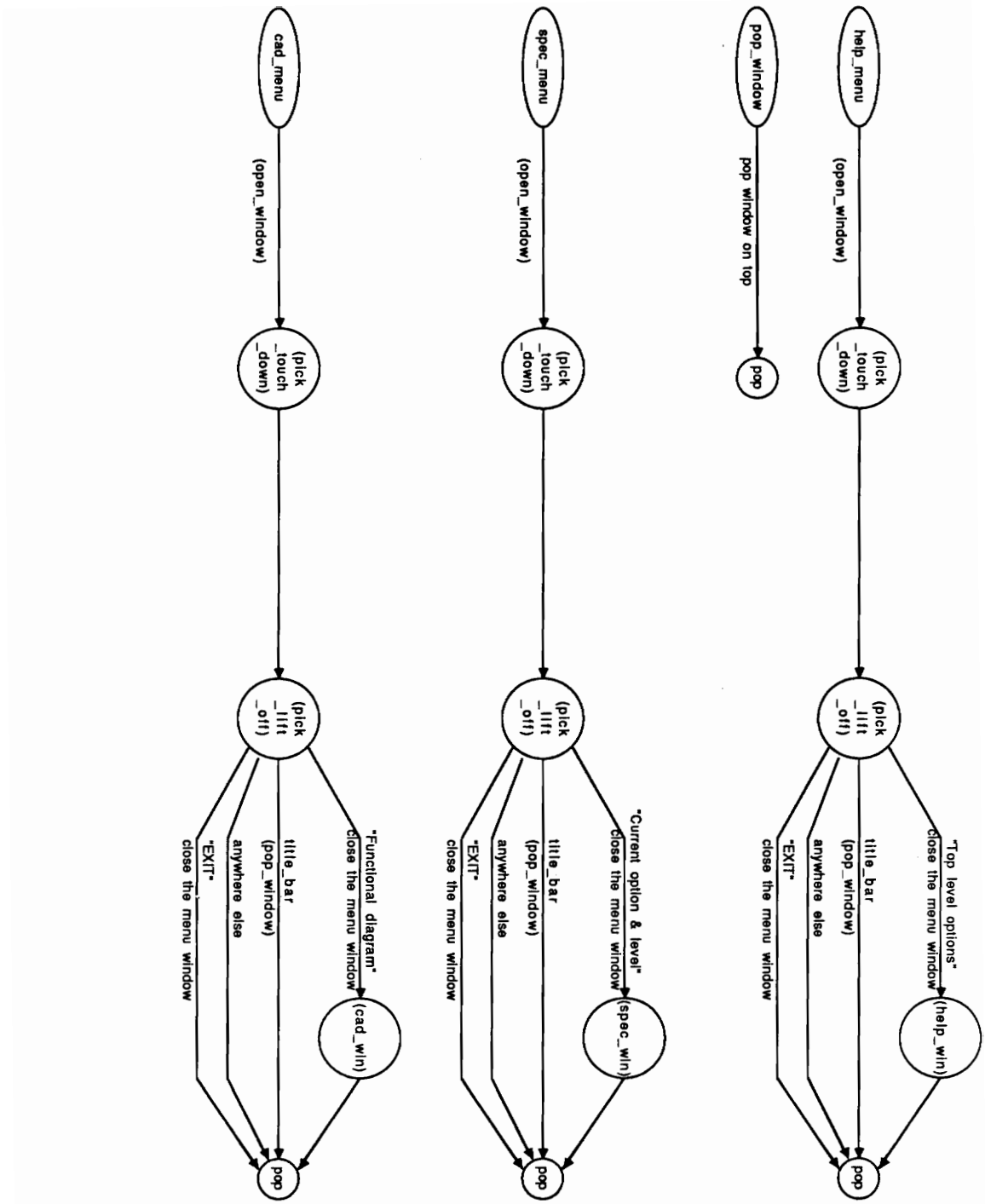
cursor_control
< <-> or < ->>
move cursor
< <+> or < +>>
pop

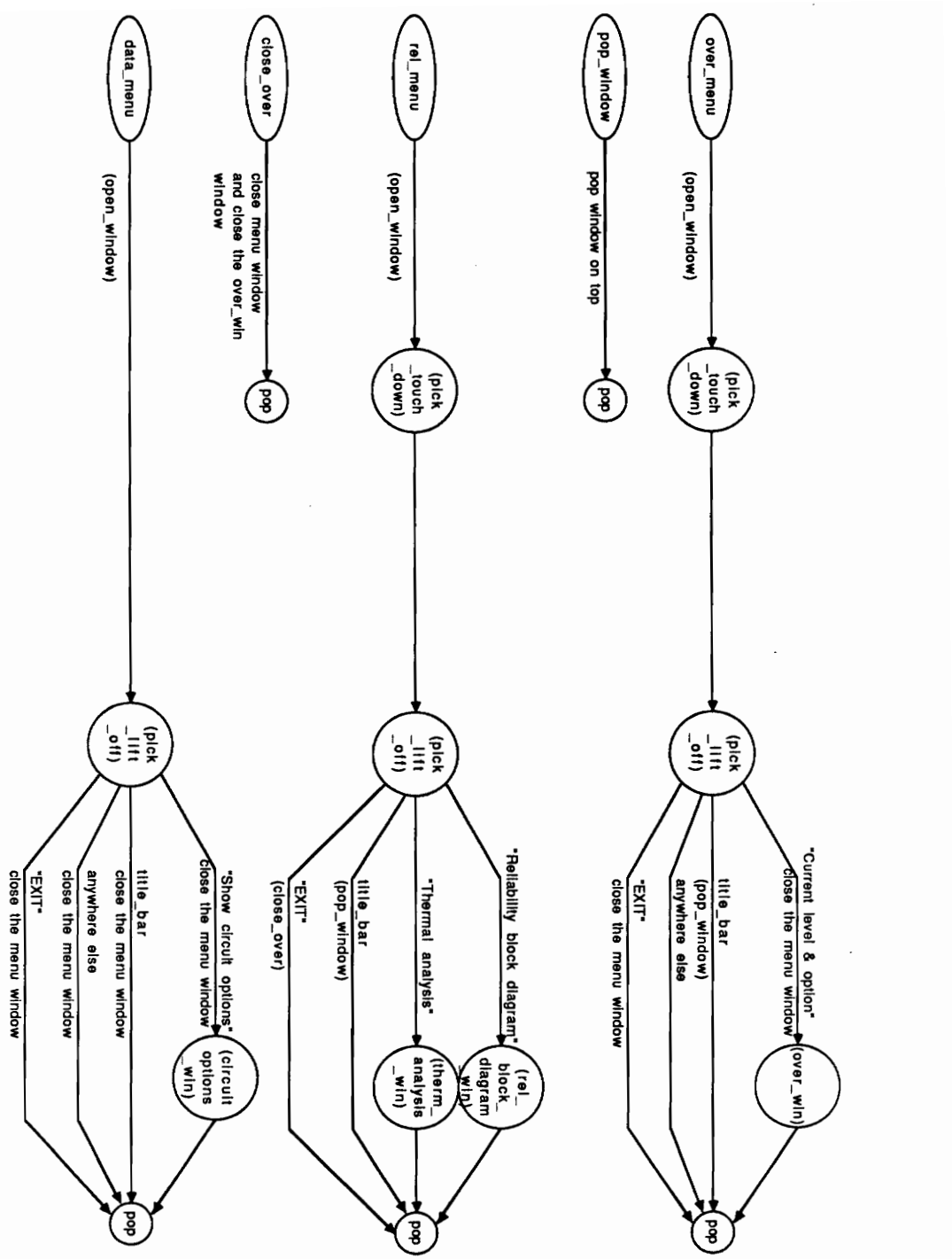
pick_touch
_down
mouse pointer
(pick_wait
_down)
pop
pick_lift
_off
mouse pointer
(pick_wait
_up)
pop

mouse movement
move the mouse pointer
pick_wait
_down
mouse
button
(mouse_
button)
button=<left> & position=down
pick_location <- mouse pointer location
anything else
pop

<left> released
button <- <left>
position <- up
<left> depressed
button <- <left>
position <- down
<right> depressed
button <- <right>
position <- down
<left> & <right> depressed
button <- both
position <- down
<left> & <right> released
button <- both
position <- up
<right> released
button <- <right>
position <- up
mouse_buttons
pop

mouse movement
move the mouse pointer
pick_wait
_up
mouse
button
(mouse_
button)
button=<left> & position=up
pick_location <- mouse pointer location
anything else
pop


help_menu
(open_window)
(pick _touch _down)
(pick _ llft _off)
"Top level options"
close the menu window
(help_win)
title_bar
(pop_window)
anywhere else
"EXIT"
close the menu window
pop
pop_window
pop window on top
pop
spec_menu
(open_window)
(pick _touch _down)
(pick _ llft _off)
"Current option & level"
close the menu window
(spec_win)
title_bar
(pop_window)
anywhere else
"EXIT"
close the menu window
pop
cad_menu
(open_window)
(pick _touch _down)
(pick _ llft _off)
"Functional diagram"
close the menu window
(cad_win)
title_bar
(pop_window)
anywhere else
"EXIT"
close the menu window
pop


over_menu
(open_window)
(plck _touch _down)
(plck _llll _off)
"Current level & option"
close the menu window
(over_win)
title_bar
(pop_window)
anywhere else
"EXIT"
close the menu window
pop
pop_window
pop window on top
pop
rel_menu
(open_window)
(plck _touch _down)
(plck _llll _off)
"Reliability block diagram"
(rel_ block_ diagram _win)
"Thermal analysis"
(therm_ analysis _win)
title_bar
(pop_window)
"EXIT"
(close_over)
pop
close_over
close menu window and close the over_win window
pop
data_menu
(open_window)
(plck _llll _off)
"Show circuit options"
close the menu window
(circuit options _win)
title_bar
close the menu window
anywhere else
close the menu window
"EXIT"
close the menu window
pop

help_win
(open_window)
(pick _touch _down)
(pick _lift _off)
title_bar
(pop_window)
anywhere else
"EXIT"
close the help window
pop
pop_window
pop window on top
pop
spec_win
(open_window)
(pick _touch _down)
(pick _lift _off)
work_area_win
(pop_window)
anywhere else
"EXIT"
close the spec window
pop
cad_win
(open_window)
(pick _touch _down)
(pick _lift _off)
functional_block
(cad_win)
anywhere else
"EXIT"
close the cad window
pop

over_win
(open_window)
(pick _touch _down)
(pick _lift _off)
title_bar
(pop_window)
anywhere else
"EXIT"
close the over window
pop
pop_window
pop window on top
pop
rel_block diagram_win
(open_window)
(pick _touch _down)
(pick _lift _off)
title_bar
(pop_window)
anywhere else
"EXIT"
close rel_block_diagram window
pop
therm_analysis win
(open_window)
(pick _touch _down)
(pick _lift _off)
thermal_parameter
(parameter _mode)
"RUN"
update over_win
"QUIT"
close the therm_analysis window
title_bar
(pop_window)
anywhere else
pop

circuit_options_win
(open_window)
(pick_touch_down)
circuit_option
display new circuit in circuit options window
(pick_lift_off)
title_bar
(pop_window)
anywhere else
pop
"ADD"
add new circuit to function_block and close the circuit_options window
"EXIT"
close the circuit_options window
pop_window
pop window on top
pop

(help_win)
(spec_win)
(cad_win)
work_input
(over_win)
pop
(rel_block_diagram_win)
(thermal_analysis_win)
(circuit_options_win)
anywhere else

window already open
open_window
pop
window not open
open window

Appendix K. Questionnaire

QUESTIONNAIRE ABOUT THE ANALYSIS TASK

Now that you have completed the human factors analysis of the interface specification and seen the other specification tools, I would like to you to answer a few questions about the analysis task and the specification tools.

On the next few pages there are some rating scales that relate to the human factors analysis you just performed. The scales are numbered 1 to 5 and labeled at each end. For each scale there is a description of the rating. Read this description carefully and circle the number which best describes your experience.

DIFFICULTY refers to how hard or difficult you found the task.

YOUR DIFFICULTY RATING

TASK COMPLEXITY refers to how complicated or complex you found the task.

YOUR TASK COMPLEXITY RATING

MENTAL WORKLOAD is the integrated mental effort required to perform the task. It refers to the depth of thinking required by the task.

YOUR MENTAL WORKLOAD RATING

STRESS LEVEL refers to your emotional reaction while you performed the task. Stress may be considered your feeling of anxiety, concern, uneasiness and uncertainty brought on as a direct result of performing the task.

YOUR STRESS LEVEL RATING

In this section of the questionnaire you will be asked to compare the specification tool that you used in the analysis session with the other two specification tools you just learned about.

Your task in this section is to rank the specification tools from 1 to 5 along several dimensions for three different components (input, output, and control flow) of the user-computer interface (see Figure 1). These dimensions and the specification tools are presented in the three tables below. For each specification tool and rating dimension combination you will put a whole number from 1 to 5 according to the following procedure.

1. Ranking is done on a scale of 1 to 5. With a 1 indicating a low ranking and a 5 indicating a high ranking. Only whole numbers can be used.

2. The ranking is made relative to the other specification tools that you have used and learned about.

3. A high ranking (5) means that a specification most closely conforms to the definition of a ranking dimension.

Ranking Dimensions

Below are the definitions of the dimensions that you will use to rank the specification tools for each of three components of user-computer interfaces. Following each definition is a short example of how you might go about ranking a tool along the dimension.

UNDERSTANDABILITY: The specification of the interface must be easily understood by the human factors expert without requiring her to have in depth knowledge of computer science or linguistics (the formal study of human language and communication).

"For example if you feel a specification is too hard to understand give it a low number ranking."

EFFICIENCY: The time and effort needed to make a specification of the interface using the specification tool must be less that the effort needed to produce the software for the actual program.

"For example if you feel it would take very little time to construct a specification using a certain tool then give it a high number ranking."

EXPRESSIVE POWER: The specification tool must be powerful enough to represent complex interfaces in an abstract and precise way.

"For example if you feel a specification tool is too simple minded to represent complex computer interfaces, then give it a low number ranking."

FIDELITY: The specification tool should be able to describe the look and feel of the real interface as much as possible.

"For example if you feel a specification tool <u>can</u> give you a good impression of what the real interface will look like, give it a high ranking."

INPUTS

Please complete the table below. Rank each specification tool along the dimensions of understandability, efficiency, expressive power, and fidelity only for the INPUT component of a user-computer interface. INPUTS are the user's actions (e.g. keyboard inputs, mouse movements, picking of objects on the display, etc.).

"For example, if you feel that a specification tool cannot in any way describe the feel of using a mouse to pick a menu item in the interface, then give a low ranking on the FIDELITY dimension".

	GTN	SFG	R-P
UNDERSTAND -ABILITY			
EFFICIENCY			
EXPRESSIVE POWER			
FIDELITY			

GTN = Generalized Transition Network
SFG = Semi-Formal Grammar
R-P = Rapid=Prototype

OUTPUTS

Please complete the table below. Rank each specification tool along the dimensions of understandability, efficiency, expressive power, and fidelity only for the OUTPUT component of a user-computer interface. OUTPUTS are the computer generated text, graphics, sounds, etc that the user sees on the display.

"For example, if you feel that an interface designer could quickly and easily describe the OUTPUTS of an interface using a particular specification tool, then give the specification tool a high ranking on the EFFICIENCY dimension".

	GTN	SFG	R-P
UNDERSTAND -ABILITY			
EFFICIENCY			
EXPRESSIVE POWER			
FIDELITY			

GTN = Generalized Transition Network
SFG = Semi-Formal Grammar
R-P = Rapid=Prototype

CONTROL FLOW

Please complete the table below. Rank each specification tool along the dimensions of understandability, efficiency, expressive power, and fidelity only for the CONTROL FLOW component of a user-computer interface. CONTROL FLOW is the component of the interface that controls when an interface changes modes or behavior. The CONTROL FLOW describes for example what OUTPUTS occur for a given INPUT.

"For example, if you feel that a specification tool can only poorly describe when and why a certain INPUT will cause the program to lose all of the users previous work, then give it low ranking on the EXPRESSIVE POWER dimension".

	GTN	SFG	R-P
UNDERSTAND -ABILITY			
EFFICIENCY			
EXPRESSIVE POWER			
FIDELITY			

GTN = Generalized Transition Network
SFG = Semi-Formal Grammar
R-P = Rapid=Prototype

You have finished the questionnaire! Please take one more look over your work. The experimenter will be with you soon. Thank You!!

Vita

William W Smith III

PERSONAL

Birth date: April 14, 1963

EDUCATION

M.S. in Industrial Engineering and Operations Research, The Human Factors Department, Virginia Tech, Blacksburg, Virginia 24061. February, 1988.

B.S. in Industrial and Operations Engineering. The University of Michigan, Ann Arbor, Michigan 48109. Dec. 1985. Relevant course work:

WORK HISTORY

Graduate Research Assistant
Human-computer Interaction Laboratory
Virginia Tech
Blacksburg, VA 24061

Analyzed USI specification tools and conducted an experiment to determine the usefulness of the tools to a human factors engineer.

Courseware Designer
Microsoft Corporation
Redmond, WA

Produced computer based tutorials for a wide range of software products. Created and structured the information display of an on-line help system.

PUBLICATIONS

Smith, W, and Williges, B. (1987). **A review of user-computer interface specification techniques to define RAMCAD interface requirements.** Blacksburg, VA: Department of IEOR, Virginia Tech, Interim Report, Contract Number F33615-87-C-0002

Smith, W, and Williges, B. (1987). **Evaluation of software interface specification tool for human factors engineering analysis.** Blacksburg, VA: Department of IEOR, Virginia Tech, Report, Contract Number F33615-87-C-0002

SOCIETIES

Human Factors Society. ACM (SIGCHI).

William W. Smith III

William W. Smith III