Technical Report
CMU/SEI-87-TR-6

User Interface Technology Survey

Technical Report

CMU/SEI-87-TR-6

1987

User Interface Technology Survey]

Software Engineering Institute

Carnegie Mellon University

Pittsburgh, Pennsylvania 15213

Foreword

The Technology Identification and Assessment Project combined a number of related investigations to identify:

- existing technology in a specific problem area to review research and development results and commercially available products;
- new technologies through regular reviews of research and development results, periodic surveys of specific areas, and identification of particularly good examples of the application of specific technologies;
- requirements for new technology through continuing studies of software development needs within the DoD, and case studies of both successful and unsuccessful projects.

Technology assessment involves understanding the software development process, determining the potential of new technology for solving significant problems, evaluating new software tools and methods, matching existing technologies to needs, and determining the potential payoff of new technologies. Assessment activities of the project focused on core technology areas for software engineering environments.

This report is one of a series of survey reports. It is not intended to provide an exhaustive discussion of topics pertinent to the area of user interface technology. Rather, it is intended as an informative review of the technology surveyed. These surveys were conducted in late 1985 and early 1986.

Members of the project recognized that more general technology surveys have been conducted by other investigators. The project did not attempt to duplicate those surveys, but focused on points not addressed in those surveys. The goal in conducting the SEI surveys was not to describe the technology in general, but to emphasize issues that have either a strong impact on or are unique to software engineering environments. The objective in presenting these reports is to provide an overview of the technologies that are core to developing software engineering environments.

1. Introduction

One of the core technology areas in which project members conducted a survey was user interface technology. This report attempts to do two things: specify an understanding of user interfaces by presenting a taxonomy that encompasses the various aspects of user interfaces, and indicate the state of the technology today by highlighting some of the major issues.

This chapter overviews topics that are important to the general area of user interface technology. Chapter 2 presents a taxonomy of aspects of the user interface that are important to software engineering environments. Chapters 3 through 6 elaborate on the taxonomy. In closing, Chapter 7 summarizes some major issues that need to be addressed in the near future.

1.1. The User Interface

The user interface is the part of the system that allows the user to enter, store, manipulate and retrieve data, and initiate commands. It enables communication between the user and the computer. Through the interface, the user gets an impression that influences judgements about the system. A well-designed user interface allows the user relatively easy access to the power of the system. Conversely, a poorly designed user interface can make even the best system seem unduly complicated, thereby rendering it virtually unusable to all but a handful of experts.

User interfaces of applications can be divided into noninteractive interfaces and interactive interfaces. For noninteractive interfaces, input must be prepared before the actual execution of the respective application. Resulting output is often logged for later printing and examination. Despite their characteristics, these applications are often invoked through interactive media such as terminals and workstations. applications with interactive interfaces take better advantage of the potentials of interactive media. They are more responsive to the needs of their users, provide more immediate feedback to user actions, and increase user productivity.

Because of their rising popularity, interactive interfaces have received attention from numerous disciplines, each contributing new insights into the strengths and weaknesses of various interactive user interfaces. Research and development efforts related to human-computer interaction have been carried out in technical and human factors areas.

1.2. Technical Contributions

In the technical area, contributions include interactive graphics, window systems, user interface management systems, forms systems, document production systems, device technology, and programming systems. These contributions have provided a number of alternative modes for human-computer interaction.

- **Interactive graphics** was the first area to employ a range of user interface techniques based on pointing devices and graphic displays.
- **Window systems** are packages, or "toolboxes," of display mechanisms. They have varying degrees of sophistication, providing such mechanisms as windows, menus, scrolling, selection buttons, forms, and dialogue boxes.
- **User interface management systems** (UIMS) separate the user interface from the application. The user interface layout and dialogue behavior are specified formally. The application program is isolated from the details of the interaction through a well-defined procedural interface. A UIMS runtime package tends to mediate the interactions at runtime.
- **Forms-based systems** provide formatting specifications for printing on preprinted forms and entering data into online forms. An obvious advantage of such an interface is that it provides for consistency across a group of users in terms of format of both input and output data.
- **Document production systems** provide user control over font styles and sizes, text formatting, placement of figures and tables, accurate cross referencing by page or section numbers, and bibliographic and editorial support.
- **Device technology** continues to introduce new input and output media and devices. Available technologies include speech input and output, and scanners that read printed text and images into the system for storage and reuse.
- **Programming systems** provide language-sensitive support through highly interactive tools such as structure editors and debuggers. Such tools enable a programmer to bypass a number of the common errors and annoyances that occur during program development.

1.3. Human Factors Contributions

In the human factors area, contributions are being made by disciplines concerned with the human aspects of human-computer interaction. Relevant research is being conducted in ergonomics, cognitive psychology, behavioral psychology, the visual arts, and human factors engineering.

- **Ergonomics**, the study of the relationship between humans and their work environment, is concerned with how the physical characteristics of the work environment, e.g., the interface between the user and the computer, affects people’s ability to perform effectively in the environment.
- **Cognitive psychology** analyzes the process of interaction in order to develop models of the perception, cognition, and motor response activities that a user performs. Such models are used to predict the user’s performance and derive guidelines for user interface design.
- **Behavioral psychology** analyzes activities of users to discover recurrent patterns and develop a model describing the behavior.

- **Visual arts** contribute to user interfaces by developing symbol systems for representing information and improving the aesthetics of graphical representations.
- **Human factors engineering** is an emerging field of engineering practice that provides both practical guidelines for system design and methodologies for evaluating the functional effectiveness of existing interactive system designs.

2. A Taxonomy of Aspects of the User Interface

Different application areas place varying demands on a user interface. The taxonomy presented here summarizes application areas and concentrates on issues related to software engineering.

A user interface has two "faces": one for the end user, and one for the implementor or applications programmer. Whereas the appearance of the applications programmer interface is largely affected by the user interface architecture and functionality of the procedural interface, the appearance of the end user interface is largely driven by hardware technology and techniques developed by utilizing this technology. The acceptance of these techniques is determined by how well they satisfy the needs of both the end user and the applications programmer. Human factors engineering draws from various disciplines to contribute improvements to the user interface.

The taxonomy of aspects of this user interface technology survey includes the topics listed below. Chapter 3 discusses different application areas and their impact on user interface technology. The remaining chapters concentrate on user interface issues related to software engineering: Chapter 4 describes user interface technology and techniques as they appear to the user, Chapter 5 discusses human factors issues in human-computer interaction, and Chapter 6 discusses implementation considerations of user interfaces, i.e., how the considerations are realized. The expanded taxonomy is on the following page.

Application Areas

- Software Engineering
- Office Environment
- Engineering Environment
- Real-Time Environment
- Mixed Media Environment

Human Factors

- Models of the Interaction Process
- Evaluation of the Interaction Process
- Pictorial Representations
- Design Principles

Technologies and Techniques

- I/O Media
    - Character Output Devices
    - Graphics Output Devices
    - Input Devices
    - Logical Input Devices
- Textual Communications
- Two-Dimensional Displays
    - Display Formatting
    - Window Systems
    - Two-Dimensional Input
    - Menu Mechanisms
    - Interaction Tasks and Techniques
- Graphical Communication
- High-Level Services

Implementation Considerations

- Architectural Models
- Development Tools
    - Toolboxes and Generation Systems
    - Languages
- Portability and Standards
    - Portability
    - Official Standards
    - Defacto Standards

3. application Areas

The wide range of application areas places varying demands on a user interface, resulting in an amazing variety of user interface appearances. application areas include the office environment, engineering environment, real-time environment, mixed media environment, and software engineering. This chapter highlights the demands each of the application areas has on user interfaces. The intent is to give the reader an impression of the similarities and differences of application demands on user interfaces. The remaining chapters of the report concentrate on the application area of software engineering.

3.1. Office Environment

The office environment refers to technologies such as office information systems (OIS), management information systems (MIS), and data processing (DP). In this application area, users often have limited knowledge of computer systems. Since there is a large number of users, training users is a major concern. In OIS, one approach to interface design has been to emulate an environment with which the user is familiar [112]. For example, the desktop metaphor has been used to describe word processing, filing, retrieving, and browsing capabilities that are essential to online office work.

Document production is of great concern to an office that generates large volumes of documents, especially reports with tables, figures, bibliographies, and cross references. Integration of documentation tools is important when there is the need to move documents among different users and different tools such as document preparation tools and electronic mail systems. Another issue of integration is the inclusion of speech as an additional communication medium, which can be in the form of stored and retrieved messages [124] or speech recognition tools [96].

Management information systems require support for multiple views of information through business graphics. Part of the viewing operations are summaries of the information. In an interactive system, changes to the information in one view is expected to be propagated. Common representations for underlying information, such as the representation for spreadsheet data as promoted by Microsoft (referred to as SYLK [82]), allow information exchanges among tools not necessarily designed together. The users of such systems, often nonprogrammers, should not need to know the terminology of the mechanisms being used to achieve appropriate views. Based on the type of data and the desired effect (e.g., relative comparison), it would be highly useful for the system to provide the most appropriate and effective representation.

Data processing deals with large volumes of information. Since such information often affects people and their lives directly, stringent checking for consistency in information is important. This applies both to interactive and noninteractive data entry as well as data retrieval and printing.

3.2. Engineering Environment

The engineering environment includes computer-aided design (CAD), architecture, and mechanical engineering. In this application area, information is best presented graphically to show products and composed components that potentially decompose into smaller components. All of these components may be interconnected in ways that virtually are impossible to describe without the aid of graphics. Such components frequently are standardized and made available to the user through online libraries. Usually, there are application-specific constraints regarding component interconnections. These constraints may range from logical consistency to layout limitations.

For engineering applications that manipulate objects in three dimensions, the use of three-dimensional graphic views has opened new avenues. In chemistry, for example, visual examination of molecular structures has yielded new insights. Currently, a variety of techniques are available for representing three-dimensional shapes on two-dimensional screens, ranging from polygon meshes to shading and use of color and intensity [41].

3.3. Real-Time Environment

The real-time environment refers to applications where time is a critical factor. Two obvious application areas are process control and simulation, (e.g., flight simulation). In process control, the user interface is expected to give the user an impression of the progress of an ongoing real world process. Thus, a large variety of sensory input devices with specific applications may have to be handled. For the user to make decisions and provide feedback to the application, information about the process and its changes must be represented properly to be communicated effectively. Today, large displays incorporating graphics, color, animation, and audio output are commonplace.

Simulation applications try to model a real world environment and process for the user. In some cases, a display screen can satisfactoryly present a colorful, three-dimensional, moving world. Other applications use additional media, often physically more realistic, to communicate the impressions of the real world. Coordinated control over these devices is required to maintain a consistent view of the simulated image of the real world.

3.4. Mixed Media Environment

The mixed media environment refers to systems such as electronic publishing, electronic music composition, imaging, and speech processing. Electronic publishing pushes technology in several ways. Digital video and audio disks are used for information dissemination with local retrieval capabilities. Online retrieval capabilities are provided to a large population through videotext and similar technologies, which are being expanded to provide data entry capabilities. Finally, computerized support for production of publications is provided.

Electronic musical composition has encouraged new techniques and technologies for entering information and for visual display [7]. Current technologies allow the composer to enter a score into the system, synthesize orchestral components, and revise the score where necessary.

If speech and image processing are to be carried out in continuous form, they require techniques for handling a high volume of data. Special techniques have evolved to manipulate and display images, and scanners can be used to input two-dimensional images and text. Speech processing systems are limited in most cases by speaker dependence, i.e., they are not capable of handling the wide range of phonetic variance due to ethnic and regional differences in pronunciation and speech production rates.

3.5. Software Engineering

Software engineering activities encompass development and support efforts throughout the software life cycle. Users in a software engineering environment include, among others, the project manager, system analyst, programmer, tester, integrator, maintainer, and technical writer. Project management tools support project planning, scheduling, estimating, control, and tracing. These tools tend to be forms and graphics-oriented — much like management information systems. Software management tools control the evolution of software through version control, system modeling, and coordination and communication between multiple users [69]. Throughout the life cycle, users enter and examine information in a variety of representations, ranging from plain and program text to formatted documents and graphical charts. Users communicate in a wide variety of languages, ranging from command languages to programming languages and graphical languages. Special tools such as formatters and syntax-based editors have an understanding of the syntax and formatting requirements. Graphical representations are not only present in requirements, specifications, and design documents, but are also applied during coding, debugging, and testing activities [84, 71].

4. Technologies and Techniques

This chapter surveys user interface technologies and techniques in terms of their pertinence to the application area of software engineering. First, hardware technologies and their logical counterparts are discussed. Then, textual dialogue techniques are elaborated. This is followed by a discussion of two-dimensional user interface and graphical techniques. The chapter closes by highlighting higher-level user interface services.

4.1. I/O Media

For a long time the form of interaction with the user was driven by the limitations of the I/O media rather than the user's needs. More recently the general availability of interactive device technology such as bitmap displays and pointing devices has made it possible to tailor the user interface functionality and representation of information to the user's demands. Interactive devices used for software development fall into several classes, each of which has characteristics that affect the type of user interaction. One classification is oriented toward the output device, whereas a second classification is centered around the input device. Over time a set of logical devices has evolved that characterizes classes of interaction techniques and found its place in the graphical kernel system (GKS) standard [61]. The remainder of this section elaborates on each of the classes of output and input devices.

4.1.1. Character Output Devices

The first two classes, hardcopy terminals and low functionality CRT terminals, are mentioned because they had a strong influence on the appearance of user interfaces, many of which are still in use today (e.g., UNIX)[1].

Hardcopy terminals, which provide a keyboard as the input device and a continuous paper printer as the output device, represent a class of low functionality terminals. Due to the mechanical nature, they are low bandwidth devices. The low bandwidth usually causes the dialogue text to be cryptic, e.g., error messages are often printed in the form of an error code corresponding to a text form that must be looked up manually. This terminal class provides single-point output; the

[1]UNIX is a registered trademark of Bell Laboratories. For a list of products that have trademarks, see Appendix A.

output location (i.e., the location at which the next output will be placed) cannot be repositioned. Line-oriented editing techniques are employed. The hardcopy printout provides a transcript of all transactions between the user and the computer, which the user can search visually.

Dumb terminals, or low functionality CRT terminals, fall into a similar terminal class. The screen is two-dimensional, but it is treated as a limited-output medium. Output is inserted in the bottom line and the remainder of the text moves up (scrolls), discarding the top line. Thus, it provides a limited-interaction transcript. Some terminals in this class permit limited editing of the bottom line, i.e., erasing characters by backspacing. The nonmechanical nature of the display hardware supports a higher bandwidth for greater interaction.

The next class of character terminals represents CRT terminals with fully addressable screens. The cursor, i.e., the symbol indicating where the output is being placed, can be positioned at any location of the two-dimensional character matrix. Such display capabilities support full screen editors with multiple windows, e.g., EMACS [121], and forms-based interaction techniques.

A fully addressable screen provides a real two-dimensional display capability. A rich set of screen editing operations is embedded in the display hardware/firmware. It includes operations such as insert line, delete line, and repeat commands. Display attributes such as reverse video or field protection can be set on a character or line basis. Some terminals, such as the Concept 100 terminal, support subdivision of the screen into regions (windows).

The American National Standards Institute (ANSI) developed a terminal standard (ANSI X3.64) for full-duplex operation, which was realized in draft form in the DEC VT100 terminal with minor differences to the approved standard. This standard did not prevent manufacturers from offering additional functionality, which allowed a proliferation of display codes in addition to the provision of emulation of ANSI X3.64 or DEC VT100 functionality. To achieve portability of applications across the range of emerging terminal types, the UNIX operating system provides a virtual terminal interface. The implementation of this interface uses a database of terminal-specific characteristics (termcap [134]). New terminals can be supported by adding the appropriate termcap to the database.

4.1.2. Graphics Output Devices

The class of simple graphics terminals represents CRT terminals with character position addressability and graphics character sets. Characters in this set provide line drawing elements that may be composed to achieve the desired effect. An example is the DEC VT102. The technique of graphic character sets provides limited graphics capabilities, which are used mostly in application areas such as business graphics. This technique is also used on many of personal computers. The use of color has not permeated software engineering applications other than business graphics for project management.

The class of full graphics terminals is represented by two hardware technologies: *vector graphics*, and *bitmap graphics*. Vector graphics dominated the CAD application area for a long time. However, with maturity and increased digital processing power, bitmap graphics became the more prevalent technology because it was more versatile. A bitmap display supports individually addressable pixels. Xerox Parc became a forerunner in using this technology in a personal computer workstation for uses other than graphics applications [12]. The bitmap display and availability of the mouse as a pointing device introduced a range of new interaction techniques, many of which were explored for practical use in the Smalltalk system [48]. The high bandwidth communication provided a means to represent information in forms closer to those normally used within particular domains. Examples are WYSIWYG (what you see is what you get) document preparation systems and symbolic object representation (icons). The dedicated processing power of such systems enables a human-computer dialogue that is more user friendly, especially for noncomputer people. The results of this work are now found in products such as the Apple Macintosh [139].

4.1.3. Input Devices

The simplest input device is a keyboard. The most commonly used layout for alphanumeric keys is the QWERTY layout, which is found on typewriters. Alternative keyboard layouts such as Dvorak have been developed, but have not found general acceptance. A basic keyboard usually includes special keys such as the escape or the control key. A lack of standards results in variances in the layout of nonalphanumeric and special keys on keyboards of different vendors. Many terminal keyboards are extended with several key sets such as a numeric keypad for efficient numeric data entry, a screen cursor motion pad, and function keys that can be programmed to execute a sequence of keystrokes or assigned by software to perform special functions. Martin [75] provides an extensive, although dated, discussion of general- and special-purpose keyboards and techniques to simulate keyboard extensions such as function keys.

A variety of devices have been developed to augment keyboard input, thereby providing a more effective means for the user to address a two-dimensional display. Such devices include the mouse, light pen, and touch pad. Other devices, such as the joystick, are a realistic means for providing analog information. The mouse has found the most acceptance in software engineering applications.

4.1.4. Logical Input Devices

The evaluation of physical input devices with regard to a logical device classification yields some useful insights. A set of logical devices has been defined [88, 41] and is supported as part of the GKS standard [61]. These logical devices support different interaction techniques such as *locator, pick, stroke, valuator, choice,* and *text*. Some of these techniques can be simulated easily by using alternative selections, e.g., the choice selection can be simulated by using picking techniques [36]. As a result, given the assumption that only one device in addition to the keyboard is available, devices such as the mouse are the most versatile. Preferences between devices within this group are based largely on motor skill demands in handling the device. An analysis of physical and logical devices and their support for different interaction techniques is provided by Foley, Wallace, and Chan [42], and Card, Newell, and Moran [26]. Foley et al. describe a systematic structure based on which devices and interaction techniques can be matched with the interaction tasks to be performed. Card et al. determine experimentally the effectiveness of several input devices for certain types of operations. Results show that keyboard commands are more effective for short distance positioning, whereas a mouse is more effective overall.

4.2. Textual Communication

Martin [75] gives a somewhat dated but still valid discussion of textual dialogue alternatives. The list includes English-language techniques, mnemonics, prompting, menu, and form filling techniques. All are centered around a command language and different ways of entering command language statements into the system.

Command languages are the means of communication, not only for the system command interpreter, but for applications as well. Command languages can take the form of natural language, limited natural language, or programming language. Natural language communication would be ideal, but it is too imprecise to be practical with current natural language processing technology. Limited natural language may suggest to the user natural language capabilities that do not exist and confuse the user. Some programming languages have been developed with a syntax and vocabulary resembling natural language, e.g., database query languages. Other command languages of the programming type use obscure syntax and cryptic vocabulary, e.g., *csh*, a variant of the UNIX command interpreter.

Many command languages, such as editor command languages, have a simple syntax for their statements in the form of object/location plus operation, or operation and object/location. Some commands may take additional parameters, which may be positional or named. In some languages, all parameters must be explicitly specified, while others assume default values that can be overridden by explicit parameter specification.

In addition to a basic command set consisting of command language statements, command languages may contain control constructs, including the notion of command procedure. Some command sets may be extended and tailored by the user, existing commands may be rebound to different names or function keys, and new commands may be introduced in the form of programs or command procedures. Such features may be supported statically, e.g., at login time, or dynamically. Commands may be interconnected (pass information) through a direct link such as UNIX pipes, through unnamed containers such as "kill buffers" in some editors or "clipboards" in the Macintosh, or through named tokens such as command language variables.

Commands can be processed in different ways, providing varying amounts of immediate feedback. For example, the user may have to type a complete command line before the system starts processing, or a more flexible system may provide command completion (upon request or automatically) and parameter prompting. An extension of parameter prompting is form filling (see section 4.3.1.).

To adapt to the proficiency of different users, command interpreters support several types of command entry. Experienced users tend to prefer interfaces with short queries and few keystrokes, relying heavily on mnemonics, function keys, and remembered sequences of operations and commands. Novice users require conversational interfaces that guide them through the task with questions to which they respond by typing in either the answers or a selection character from a displayed menu. An individual user can be a novice with one application and, at the same time, an expert with a different application. Even within one application a user can be an expert in a subset of commands and a novice in the remainder [137]. The proficiency level of a user is augmented by frequency of use. An experienced user may use certain features of an application only occasionally and, therefore, desire different support for these features than for constantly used features.

The system may attempt to deduce the level of expertise or let the user decide. Time affects the level of expertise; in other words, nonuse of an application may regress the user from expert to novice. The user may be forced into one dialogue style based on the system-deduced level of expertise or be allowed to switch freely between styles.

Online help facilities are useful to assist the user at any time. Effective help mechanisms can be customized to meet the needs of the particular user. For example, an experienced user may need help in remembering a seldom used mnemonic, and hence require only a list of the contextually applicable mnemonics. On the other hand, a novice user may require a guided tour of the application.

Commands may be organized into context-sensitive sets. For example, on the Xerox Star fileserver commands are grouped into a base set and several functional sets, one of which can be accessed at any time. Such an organization helps manage large command sets, but introduces operational modes, which can increase the complexity of the dialogue model [131].

Command interpreters have varying degrees of tolerance for user errors. Simple text entry errors cause one of several responses: the user may be informed by a cryptic message and asked to retype the command, the user may be able to recall the command and edit it using standard editing operations, or the system may attempt to correct the most common errors (typing and spelling) and request confirmation from the user (DWIM or "Do What I Mean" [129, 109]). The command editing capability may be enhanced to recall any of the previously executed commands. Some erroneous commands may inadvertently invoke a command that possibly destroys information. Inadvertent execution of destructive commands can be safeguarded against through an undo function, frequent checkpointing, and a request for confirmation by the user.

4.3. Two-Dimensional Displays

The availability of randomly addressable two-dimensional displays started an era of two-dimensional communication. These displays were first available in the form of character CRTs augmented with a graphic character set, and vector graphic terminals. Recently, bitmap terminals have proliferated the market of computer workstations. Together with the availability of pointing devices, these displays have encouraged new forms of user interaction. These include form filling systems, WYSIWYG systems, window systems, pointing as input technique, and a class of dialogue mechanisms.

4.3.1. Display Formatting

Form filling systems have become quite popular as a user friendly form of interaction, even on character terminals. The user is presented with a form on a screen area that consists of explanatory information and fields in which data can be entered and modified. When a form is brought up, the fields may contain default values. The user may be required to fill in the fields in a particular order or indicate the appropriate order. As the user enters information, the form system may run some consistency checks to ensure the correct type of data. In some cases, constraints on the value range are also checked.

In general, the user indicates that a form is completed by issuing an explicit command. In some systems, completion of the last field indicates completion of the form. In addition to being used in the data entry area (e.g., airline reservation systems), forms systems have been used as command interfaces for programmers to help reduce the complexity of command entry, as in the Software Productivity Facility by IBM.

Syntax-directed editors, a popular research area since the late seventies [127, 37, 40], can be considered a specialized type of form filling system. They also have been used not only as information entry facilities, but as the user interface of a programming environment [54] and other applications [90].

A more general form of text formatting on the screen is provided through interactive document preparation systems. In such systems, the user sees the document on the screen in the form in which it will be printed — to be exact, the displayed form is usually an approximation due to differences in display density. Display capabilities of such systems include filling and justifying text, columnizing, using multiple fonts, and mixing text and graphics. Examples of such systems in product form are Xerox Star [112], Apple MacWrite/MacPaint/MacDraw [72, 74], and InterLeaf [60]. Systems such as Etude [55] are exploring models of functional division of formatting capabilities between the application and the user interface. Meyrowitz and van Dam [79] and Furuta, Scofield, and Shaw [45] provide an extensive survey of text editing and document production systems.

4.3.2. Window Systems

Some two-dimensional displays can be divided into several independent display areas called windows. Different applications can use different windows to interact with the user concurrently. One application can use multiple windows to organize the information it intends to display. In contrast to early experimentation with windowing through the use of multiple displays [123], screen partitioning permits one display to support multiple virtual display areas. Each of these virtual displays may have a different functional behavior. Three types of windows are distinguished: terminal emulation windows, typescript windows, and bitmap windows. Since multiple windows share one physical screen, screen area is a scarce resource. Several techniques of arranging windows have evolved for allocation of this resource and are discussed in this section.

Terminal emulation windows simulate the behavior of a character terminal, usually a DEC VT100 or an IBM 3270 terminal. The windows may be repositioned on the screen by the user. Windows come in several types. The size of the window may be predefined with the same display dimensions as the terminal being emulated. The size may be fixed (as in AT [8]) or changed by the user. In the latter case, the window system adjusts the font size to fit the display dimensions into the available area [110], shows the display area of the emulated terminal only partially (as in MacTerminal [73]), or automatically wraps lines.

Terminal emulation windows have several benefits. For example, they allow programs designed for character terminals to run on workstations without modifications to the display part of the application. Also, they allow the window to act as a terminal to another machine [110]. In case of a network connection, appropriate network services must be in place to establish connections and run network protocols. Thus, the user is able to connect to different machines from the same workstation by using multiple terminal emulation windows. The disadvantage of terminal emulation is that the capabilities of bitmap displays are not fully utilized.

The second window type, typescript windows [110, 73], combines properties of hardcopy terminals with those of CRTs. The display area is assumed to extend beyond the vertical dimension of the window. Input is entered at the bottom of the window and displayed information is scrolled upwards. Information that moves off the window is still accessible, providing a history of the interaction. Through special window commands, the user can move the window back over this transcript of interaction. In general, a special command is provided to resubmit any line (or input line) of the transcript as input to the window. Transcripts, however, are not provided without cost; storage must be provided for the transcript content as part of the window. For that reason, the length of the supported transcript in some window systems is limited to a predefined size. In general, typescript windows are limited to displaying text.

The third window type, bitmap windows, makes available the full functionality of a bitmap display. In general, the application can operate in a virtual area that is larger than the window. The window system performs the necessary clipping, i.e., limits the actual display to the window area. The window system may provide storage for maintaining the content of the window or the virtual area. This permits the window system to refresh the display and, in the latter case, support scrolling over the virtual area without application intervention. The storage cost may be quite high due to storing full bitmaps unless more compact representations are employed.

There are three basic techniques for arranging windows on the screen: overlapping, nonoverlapping, and tiling. Overlapped windows are like pieces of paper that can be arbitrarily sized and cover each other. The result is that the display space provided by the windows may be larger than the display space of the physical screen. However, since windows may be (partially) covered, clipping mechanisms need to take this into account. Furthermore, the window system must maintain the content of the covered areas in special storage unless this responsibility is passed to the application. As the number of windows increases, some windows can become hidden under others. To access a hidden window, the user may have to perform one of several actions, depending on the system being used:

- close or shuffle other windows, as on the Macintosh [139], in Smalltalk [49], or on the Apollo [3];
- access it by name through menu selection, as in Andrew [110];

- access it by command, as in EMACS [121];
- accessible it by icon, as in Sapphire [85] or Microsoft Windows [70].

In the nonoverlapped windowing technique, windows are always completely visible. New windows can only be placed in previously unused screen areas. As a result, the user may have to rearrange and resize existing windows. It is difficult to utilize all of the available display space.

The tiled window approach differs from nonoverlapping windows in that the screen area is always fully used. In order to make room for a new window, existing windows are resized and/or reshaped by the window system. Depending on the placement algorithm used, this results in a high level of activity on the display screen, making it difficult for the user to keep track of window locations.

Various methods exist to determine the placement of new windows [110, 70]. They often make it difficult for the user to arrange a set of windows in a certain way. Few window systems allow the user to specify the layout of a set of windows such that the layout can be automatically set up, e.g., after a login [119, 3].

It is desirable for users to group windows and treat them as one working context, similar to the notion of a desktop. It should be possible for the user to set up several contexts, not all of them fully visible simultaneously, and interactively switch between them. An alternative to window grouping is the ability of an application to divide a window into subwindows [85, 49]. However, subwindows often have limited functionality. Cedar [130] provides a general concept of windows called viewers that can be hierarchically organized.

Windows can have optional title lines, permanent and pop-up scroll bars, thumbing facilities, window movement, and sizing functions by command or pointing device. Users may be able to transfer information between windows through cut and paste (delete and insert) operations. For example, in the Xerox InterLisp environment a user can copy a piece of Lisp code received through electronic mail into the InterLisp window and execute it [128]. Some systems, e.g., Macintosh MacWrite/MacPaint, support exchange of formatted text and graphics as well as plain text. In such systems, the underlying applications, if different, must use a common representation for the information structures to be exchanged.

4.3.3. Two-Dimensional Input

The two-dimensional display and the ability to refer to a random position on the screen (via keyboard command or pointing device) provide new ways to communicate. A variety of physical input devices exist. Device-independent graphics packages such as GKS [61], and works such as Foley and van Dam [41] and Newman and Sproull [88] use logical input devices to achieve physical device independence. The articles by Foley and van Dam [41] and Newman and Sproull [88] discuss the relationship of logical input devices to physical input devices as well as to different interaction techniques (see also section 4.1.4).

One of the more popular pointing devices is the mouse, which can have a number of buttons commonly ranging from one to four. Mouse buttons can be considered special function keys that suggest execution of functions related to the mouse/cursor position. As can be seen from existing systems, functions performed by a mouse with multiple keys are often simulated by a one-button mouse or keyboard keys. The following set of examples illustrate this point.

Direction can be indicated with a four-button mouse where the buttons act as cursor keys (Perq), with mouse movement (Macintosh/Frogger), or with cursor keys on the keyboard. Scrolling direction can be indicated with one of two mouse buttons (Perq/Pepper), by clicking in the appropriate area in the scroll bar (Xerox Star), by function keys, or by menu. Selection of more than one item can be performed by holding a mouse button down. A selection can be extended by using an alternate mouse button (Xerox Star) or by using a key in addition to the mouse button (Macintosh). An "open" or "activate" operation on an object can be performed by double clicking (quickly pushing the mouse button twice (Macintosh, Interleaf)), by menu selection (Macintosh, Interleaf), or by object selection and invocation of special function keys (Xerox Star). As can be seen from the examples, in many cases context information such as location or time (as in button hold, click, and double click) is used to enable one-button mouse operations. The different realizations of functions require different motor skills (amount of motion and use of one vs. two hands).

Some systems allow mouse driven interactions to be executed by the keyboard as well. Apollo/DMEdit, Perq/Pepper, and Sun/EMACS are examples of text editors where cursor motion can be done from both the keyboard and mouse. In the case of Pepper, the functions are bound to the keyboard in such a way that the hand operating the mouse also serves the keys assigned to the mouse equivalent functions. Left versus right handedness needs to be considered in such a case. Other systems (Macintosh) allow commands to be invoked by function key or by menu. Unfortunately, systems that support full functionality through function keys and menus are rare.

The application cursor can be bound to the mouse location or decoupled from the mouse (for example, for text entry). In the first case, the application cursor position (in general, representing the input position) is changed by simple mouse movement. Accidental mouse movement misplaces the application cursor. In the second case, the application cursor is not affected by accidental mouse movement, but must be placed by explicit operation, e.g., pushing a mouse button. Similarly, the active window (the window receiving the current input) can be determined directly by the mouse location — potentially causing unintentional window activations — or by explicit operation such as a mouse click. For applications in which the cursor directly tracks the mouse location (simulation programs), it may be desirable to have the application cursor recognize the window boundary as a hard boundary. In this case, a special operation is necessary to permit window boundary crossing.

Mouse-invoked operations are often sensitive to context and location, which provides more flexibility. Different regions of a window may permit different command sets to be called by pop-up menu. For example, in Andrew [110] the mouse location in a window header refers to window system commands, whereas application commands are available while the cursor is located within a window. Different icons for the mouse cursor give the user feedback about context changes. The availability of a menu may depend on the state of the application (e.g., file saving operations in Interleaf [60]). On the Macintosh a different approach is taken for command menus. application commands can be found in a global menu set, which resides at a fixed location on the screen. The content of the menu set changes according to the active window/application. Many window systems support direct invocation of commands in a context-sensitive manner. Relative positioning operations such as horizontal, vertical, and pagewise scrolling, and absolute positioning operations such as thumbing (pointing to a specific location within a document), sizing, and motion of windows are realized by making use of location context.

4.3.4. Menu Mechanisms

A menu allows the user to select from a standard set of objects or operations. A variety of user interface toolboxes and user interface management systems provide menu mechanisms with different degrees of functionality [5, 4, 49, 9, 57, 60, 112, 85].

Menus can be visible permanently or temporarily. Permanent menus tend to be represented as a set of buttons, possibly representing the function symbolically. Temporary menus usually are organized as a row of textually labelled entries. They become visible on top of other windows under user or application control (referred to as pop-up or pull-down). They can appear in a fixed place, or near the mouse cursor location. Menu entries can be text or icons or both (all these forms are found on Macintosh). The size of the menu window can adjust to the number of entries or can be fixed. There can be an upper limit on the number of entries, or the menu content can be scrolled or paged. The set of menus and their contents may be frozen, or they can change at runtime. A subset of the entries may be active at any time.

Temporary menus are called up and selections are made in one of several ways. Some systems display the menu when a mouse button is pressed. While the mouse button is held, the mouse is tracked for menu entry selection by highlighting. Releasing the mouse button causes the currently highlighted entry to be selected. Release outside the menu window aborts the menu selection. In the latter case, the invoker of the menu function must be able to handle this exceptional condition, i.e., no selection. Other systems display a menu upon mouse click or function key invocation. The user can then select an entry through a separate click operation. Invoking a function key is usually for menus with multiple selections. Multiple selections may be indicated by clicking multiple entries and terminating the selection process by an explicit "close menu" command, or by repeatedly calling up the menu and adding to the selection. Selected entries are indicated by a check mark or by highlighting. Sometimes the selection state is remembered between menu call-ups. It provides a context for extension of selections or default selection based on previous actions. In such a case entries may have to be unselected explicitly, or a reset operation is provided. For menus with state memory, single entry selection can be enforced through so-called *radio button* behavior, which means that selection of one entry removes a previous selection.

Menu entries may be grouped and organized into menu hierarchies. All levels of a hierarchy may be visible at once, or they may have to be navigated with only the selected path visible. In some systems, a special entry in the top level menu permits the previous selection to be repeated [110, 60]. This may be carried out by immediate reselection of the final entry or by time delayed walking through the menu hierarchy.

Menu selection may be in response to a question or a message. The text may appear separately or together with the menu entries in a so-called dialogue or alert window [5] — also referred to as prompters and confirmers [49]. For ranges of numerical values, selection can take a different form such that the set of discrete values is mapped onto dials of various shapes and forms. The user can select/set a value by absolute reference or change it relative to a given value. The user may interact with a window that combines the above selection mechanisms with fields for text entry.

Because the multiplicity of alternatives is overwhelming, there is a need to classify the techniques. The techniques have to be appropriately mapped to the interaction tasks to be performed by the user. The Descartes Project [113] examined the menu as an example technique and used suitable abstractions for interactive communication to determine the role of menus in dialogues. In the process, a design space was determined that explains the variety in menus. A more general treatment of interaction tasks and techniques is presented in Foley, Wallace, and Chan [42].

4.3.5. Interaction Tasks and Techniques

Interaction tasks provide a user-oriented classification rather than the system-oriented classification of logical devices. Interaction tasks are primitive action units performed by users. They form building blocks for complex interaction tasks and complete interaction dialogues such as those found in user interface toolkits, e.g., the Macintosh toolkit [5], and user interface management systems [132].

Foley et al. [42] define six types of interaction tasks:

1. Selection — choosing from a set of alternatives, e.g., by menu or abbreviated name typing;
2. Positioning — indicating a location, often in the context of a command, to place an entity;
3. Orienting — specifying an entity's direction, e.g., by rotation;
4. Path — a sequence of positions and orientations ordered over time;
5. Quantifying — specifying a value (usually in a given range) through or indication on a dial;
6. Text — entry can be carried out by typing, or by character or word selection from a menu.

In addition, they identify four types of control tasks, i.e., tasks directly modifying displayed objects:

1. Stretching — elastic techniques such as those used to construct lines, rectangles, and circles;
2. Sketching — freehand drawing with the locator acting as a brush or pen;
3. Manipulating — moving objects, e.g., placing them in new positions on the screen (also known as dragging) or scaling an object;
4. Shaping — changing the shape of a smooth curved line or surface.

Constraints can be applied to both interaction and control tasks. One common form of constraint applied to orientation, path, and stretch tasks results in enforcing horizontal/vertical lines or circles. Grid and gravity are two forms of constraints that affect positioning and other tasks that make use of positioning.

Each of the tasks is carried out by interaction techniques. Examples of selection techniques are menus or command typing. For each technique, parameters provide further distinctions. For example, a menu is a selection technique. Menus can be hierarchical, entries can be ordered, or default selections can be provided for menus. Menus can pop up or be pulled down; they can appear in fixed places or close to the cursor.

4.4. Graphical Communication

The August 1985 issue of *IEEE Computer* was dedicated to visual programming, i.e., the use of graphics in the context of programming. Grafton and Ichikawa [51] define the term and survey work in the area. A second article by Raeder [98] surveys current graphical programming techniques and compares various graphical programming systems.

Graphics capabilities permit new forms of communication, of which we consider four types: *iconic, graphical viewing, graphical programming, and animation*. In addition, the user may communicate to the system textually while the system produces graphical representations. In this case, the user is provided with a command language with graphical operators. An example is PIC, a language for drawing simple figures [67]. Drawings specified in PIC are fed through a document processor (Troff) and can be viewed as printout or on a workstation screen. Other examples are extensions to programming languages to include graphical operations [135, 81, 108].

Iconics refers to the facility for attaching a symbolic representation to an object or entity. Xerox Star and Macintosh are good examples of an iconized user interface. Icons may be defined for each object or for classes of objects. In the latter case, a name may be included in the icon to distinguish members of a class. Icons act much like menu entries in that they are selectable. Some menu systems support an iconic representation of their entries (e.g., Macintosh). When created, icons are placed by the system or by the user. Users may place icons in arbitrary locations (Macintosh), or into a predefined grid (MS-WIN). Some systems provide an operation that organizes icons into grids and eliminates gaps (e.g., the Macintosh clean-up operation).

Movement of icons may have different semantic meanings depending on the destination location. Within certain regions, it may just relocate the physical display object. In other cases, it may move or copy the underlying object. On the Macintosh, for example, files are moved by moving an icon from one folder (directory) to another, but files are copied by moving the icon to a folder on a different disk. Moving an icon on top of a trashcan icon deletes the underlying object (Macintosh), and moving an icon onto a printer icon prints the underlying file object. Printing on the Macintosh and deleting on the Xerox Star are executed by either function key or menu.

Some systems permit the user to define iconic symbols and attach them to existing or user-created objects. Such systems are user extensible. An interesting example of an extensible application with an iconic user interface is Filevision, a visual database, on the Macintosh [83].

Iconic representations are attractive because they permit the application implementor to present to the user a view that more closely reflects his mental model of the application. However, it is difficult to appropriately apply this technique and generate a semantically consistent user interface behavior. This is illustrated with an example on the Xerox Star. Icons (and the objects they represent) can be moved and copied. Moving means removal from the source location. However, if the destination is the printer, the move operation is translated into a copy operation to avoid an unintentional deletion of the icon (object). Such action makes the user insecure because he is not sure when such a reinterpretation takes place. Alternatively, the system could prohibit the operation or ask for confirmation as a protective measure. Graphic functionality of a display permits graphical viewing of various classes of information. One class that can be viewed effectively is quantitative information such as business graphics. Such representations for numerical data, including bar charts, pie charts, and histograms, are highly effective for summarizing large amounts of data and their relations.

Graphical representations of structures are also effective for describing the complex interdependencies in the structure and at different levels of abstraction. Graphical representations also can be used as a basis for navigation. The effectiveness of graphical representation is apparent from the popularity of graphical techniques in requirements, specifications, and design methods, where data structures, data flow, and control flow can be represented graphically [122, 107, 126, 86, 117]. To date, a number of interactive systems have been built to support graphical viewing and navigation through program code and documentation in software development (PV [16], PECAN [104]), program data structures in debugging [128, 84], and directory structures (Perq [119]).

Graphical programming uses the ability to arrange graphical symbols and interconnect them to create a graphical representation, or view, of a program. Such a graphical representation is expected to have well-defined syntax and semantics and is translated into other representations. In this respect, graphical programming is similar to computer-aided design (CAD). In an interactive system, the user is informed of inconsistencies during entry of the structures. The interactive approach gives the user more immediate feedback and can aid in determining appropriate connections (e.g., PECAN [104]). In more common systems, the user enters the structures with a graphics editor, and the consistency of the structures is checked by invoking a separate tool (e.g., DesignAid [30] for data flow diagrams). The capabilities of a graphical programming editor include basic operations as well as elastic connections, layout cleanup of objects and connections, grouping of objects, and hierarchical organization of the graphic structure.

Animation is the ability to show changes over time. Graphical animation can show such progress over time effectively. One way is to visualize the time axis and generate a time plot of values. Another way is to animate the change itself in slow motion such that the user’s eye can perceive the change. Animation can be used effectively to illustrate the actions of a program, thus helping the user to understand and "feel" the actions of an algorithm, and to recognize bugs and logical inconsistencies by demonstrating changes to data structures or progress in control flow [16, 71]. Animation has also been used to monitor performance, such as network traffic and program execution profiles. Finally, animation is a tool for visually displaying the execution of simulation models.

Graphical simulation has been common practice in the artificial intelligence community for job-shop type simulations and is available in product form (e.g., Knowledge Engineering Environment (KEE) from Intellicorp, Knowledge Craft from Carnegie Group Inc.). Though graphical simulation is not yet an established tool in software engineering, it is starting to raise interest in areas such as modeling telecommunication applications [78]. Functions expected in the user interface for an animation tool include controls like those on video recorders for display speed, and reverse animation where possible. Other functions will permit the user to initialize the animation/simulation environment and change it while animation is suspended.

4.5. High-Level Services

Several higher-level concepts in dialogue management are becoming available. They are the *desktop* metaphor, *user profiling*, the notion of *browsing*, the ability to provide *multiple views*, and *transparent dialogue* throughout a *network* of workstations.

The desktop metaphor was introduced by researchers at Xerox Parc and appeared for the first time as a product in the Xerox Star office system [112]. The screen represents a desk on which the user finds a variety of objects common to the office environment shown as icons. They include inbox and outbox for electronic mail, file drawers and folders for filed documents, printers, etc. Objects can be opened, i.e., their contents examined, in overlapping windows. Thus, a desktop defines a working environment for a particular kind of user. A user can potentially have more than one desktop defined and alternate between them, which is similar to the way people alternate between physical work environments in the course of a normal workday.

It is interesting to note that systems with an "advanced" user interface (systems that make use of windows, menus, icons, and forms) lock the user into one form of interaction, such as the Xerox Star system [112]. This helps to make the system learnable, but limits effective use of it by some user groups such as keyboard-oriented users. Some systems (Apple Macintosh) [139] provide keyboard alternatives for some of the frequently executed commands. These are not easily extended by users. Other capabilities of command languages, such as background execution of programs or interactive definition of command sequences as new commands, often seem to be ignored for the sake of this "new" technology. Such features are sometimes desirable for sophisticated users to enhance the user interface of an application.

User tailoring or profiling is the ability of the user to adapt the user interface of applications to his needs. This means that the user interface and the applications have parameters that the user can set to determine certain aspects of the interaction. Examples of parameters are keybinding of functions, different modes indicating forms of mouse tracking or editing (e.g., explicit vs. implicit insert mode in EMACS), or preference for menu driven interaction vs. function key interaction. Some capabilities enabled by parameters can be used to simulate user interface concepts that may not be supported directly. For example, the ability to specify a script indicating a set of windows and applications to be started when logging into the system can have the effect of a desktop to a limited degree.

User tailoring raises several questions, for example, what is the desired degree of tailoring? The

Xerox Star system supports only minimal variations. On the other hand, the UNIX system provides a large degree of freedom. Tailoring is in potential conflict with uniformity and portability of users. It permits one user to tailor the environment to his needs, but makes it hard to interact with other users about the use of the system and provide general help. Tailoring is used to overcome shortcomings (as perceived by the user) in the application's user interface. Is this the right solution to overcome these problems?

The notion of browsing has been promoted by a variety of groups (e.g., ZOG information network [43, 44], document browsing [140], the Magpie Pascal environment [111]). Smalltalk [49] demonstrated an effective way to use the bitmap display for quickly browsing through classes in a hierarchy. The Smalltalk browser is an integral part of the Smalltalk system. In a browser, the user is presented with a root into a hierarchy or network. From this root the user can make menu selections to follow a link. The selected entity and its links are displayed in a separate area. As the hierarchy is of fixed depth in Smalltalk, a set of predefined windows works effectively. For arbitrary depth hierarchies and network structures, the selected path can be shown as a name path or graphically in an orientation map, and the last N (for fixed N) visited entities may be directly visible.

Multiple views allow the user to examine information through different representations and from different angles. This notion of multiple views has been common in database systems for some time [1], and more recently in interactive business graphics and documentation systems [60] that allow the user to have different graphical views of a set of data. Such data representations may be displayed simultaneously. In some cases, the user may modify information in more than one of the views and have the changes automatically reflected in the other views. MacProject and Jazz for the Macintosh are examples where multiple views are maintained simultaneously by the system. In the programming area, two examples are syntax-directed editors and PECAN. Syntax-directed editors support several textual representations for the same internal representation (through use of multiple unparse schemes [77]), and PECAN is the most prominent example of a programming environment supporting multiple views [104]. The general problem, however, of consistently maintaining information through modifications according to several views has not been solved.

User interface systems can provide network-transparent dialogue. The window system Andrew [110], running Berkeley 4.2 UNIX on Sun and DEC MicroVAX workstations, permits windows to be attached to processes on other machines. This allows the user to interact with shells (system command interpreters) as well as with applications on different machines in the same way he would on the local machine. The user can move or copy text within a window or between windows. Text produced as output in a window on one machine is easily copied through cut and paste to a shell window on a different machine and executed as a command. The user interface system provides all the necessary functionality such that the applications are not aware of the network.

5. Human Factors

Although technology-driven development produces powerful systems, such systems tend to contain numerous "gadgets" that may not provide the most effective means of interaction for a particular application or user. For example, menu technology for command invocation may increase the effectiveness of some users, yet decrease the effectiveness of other users. For a typist concerned only with text entry, efficiency is severely handicapped if certain functions, such as type style selection, can only be done by menu. Thus, the match among technology, application, and user is critical to the overall effectiveness of the system.

The term ergonomics has been given to research that studies work and working environments. Ergonomics stresses the central role of human physiology in the design of equipment and workplaces. The considerations include design features of the individual workspace, the type of work the individual is expected to perform, and the methods by which the work is most efficiently executed.

Human factors engineering is closely related to ergonomics. It is a field of engineering practice that offers both practical design guidelines and evaluation methodologies that help computer designers with their products. It focuses on the user aspects of human-computer interaction. It directly applies knowledge and methods that have been derived from the human sciences such as psychology, physiology, and the arts. Its working knowledge is enriched by the more informal and subjective knowledge of practical experience from people who design, build, and use systems.

Card, Moran, and Newell, members of the Applied Information Processing-Psychology Project at Xerox Parc, were pioneers in applying methods and insights from cognitive psychology to the practical design of computers [26]. The range of human psychology activities spans perception, performance, memory, learning, problem solving, and psycholinguistics. Card et al. envision subfields in the application of human psychology to information processing such as user sciences, cognitive ergonomics, software psychology, cognitive engineering, and artificial psycholinguistics. The Technology Identification and Assessment Project drew on work from the human factors area that deals directly with user interfaces. For information regarding learning and problem solving, references are given for the relevant literature.

A variety of introductory and survey materials are available covering various aspects of human factors. Burch [17] provides a recommended reading list on computer ergonomics and user friendly design. Ramsey [99], Card et al. [26], and Shneiderman [115] provide extensive reviews of psychology in human factors engineering. A more recent literature list of the field is available in the SIGCHI Bulletin [63] and as part of an article by Foley et al. [42]. The field has also received attention through a series of workshops and conferences organized by various professional organizations such as ACM SIGCHI, the Human Factors Society, and the IFIP working group 6.3.

The remainder of this section on human factors is structured as follows. First, different models of the human aspect in the interaction process are discussed. Then, methods and experimental results of evaluating different levels of the interaction process are presented. This is followed by a treatment of human factor considerations regarding pictorial representations. The section concludes with a discussion of design guidelines that are based on "words of wisdom" as well as input from various studies.

5.1. Models of the Interaction Process

Various studies have attempted to understand, analyze, and measure different aspects of human-computer interaction. Buxton [20] discusses two classifications into which those studies can be categorized, and proposes a refinement for each of the classifications. Both classifications were developed from a language-oriented view of the human-computer dialogue. The first classification, initially introduced in Foley [41], takes a language view of the user dialogue and proposes four layers: *conceptual*, *semantic*, *syntactic*, and *lexical*. The second classification, proposed by Moran [80], tries to address the device level of communication in more detail. It divides the problem area into a *conceptual component* composed of a *task level* and a *semantic level*, a *communication component* composed of a *syntactic level* and an *interaction level*, and a *physical component* composed of a *spatial level* and a *device level*. Buxton proposes a refinement of these classifications by adding the *pragmatic level* to better address spatial issues of the dialogue.

Card et al. [26] take a cognitive psychology view of human-computer interaction. They describe the model of a human processor as consisting of a *perceptual system*, a *motor system*, and a *cognitive system*. Human performance is determined by perception (reading rate), motor skill (key layout and typing efficiency), simple decision making (yes/no answers), and learning and retrieval (memory of acquired information).

Card et al. also developed a model to study the interaction process between humans and computers. The model, called GOMS, consists of *goals*, *operators*, *methods* for achieving the goals, and *selection rules* for choosing among competing methods. They use this model to analyze the human-computer interaction process for the application domain of text editing.

Norman [89] proposes a similar division of the process into four stages: *intention*, *selection*, *execution*, and *evaluation*. Intention reflects a set of goals in the user's mental model. Selection is the mental selection of actions or methods for achieving the goal. Execution is the physical act of entering information into the computer. Evaluation is the examination of the feedback as part of the result of execution to determine further activity in the process. Norman analyzes the four stages to determine the need for different support at different times in an interactive session. He points out that each interaction technique provides a set of trade-offs. The choice among the trade-offs depends on the technology used, the class of users, and the goals of the design. The choice has to be made with a global perspective, because the optimal choice for one stage may not be optimal for another.

Riley's work [105] elaborates on the intentional stage of the above model. Formulating the intention and making a selection requires knowledge of the actions available from the system, and planning for achieving the desired goals. Riley suggests that a planning framework is useful as a basis for developing general principles for instructing, designing, and evaluating features of an interface. This conclusion was drawn by examining the user's mental model of a process, such as text editing, and comparing it with the conceptual structure of the system image. It was noticed that mismappings between the conceptual structure of a command and the user's mental model could explain the difficulties in learning to use a system.

5.2. Evaluation of the Interaction Process

In a special issue of *ACM Computing Surveys* on "The Psychology of Human-Computer Interaction" [80], Moran points out that there are many dimensions along which user behavior and the effects of the user interface can be evaluated. These dimensions include:

- functionality of the user interface,
- ease of learning,
- time it takes to perform a task,
- error sensitivity, and
- quality of the resulting output.

Trade-offs must be made between the dimensions, and these trade-offs may differ with the type of user. For example, ease of learning is more important for a novice user than performance time. In general, the user interface should reflect the user's conceptual model of his task as close as possible. Often, however, the user interface is provided as an afterthought, and the functionality of the actions is driven by the implementation of the application rather than the user's conceptual model.

Considerable data is available for the ergonomic aspect of keyboard layout and design and on the design of display terminals regarding such items as display contrast, refresh rate, character size and shape, and screen orientation. Ergonomic standards exist in Europe for display terminals and the working environment, and standards are being contemplated in the US. For a discussion of the full spectrum of issues, the reader is referred to Cakir, Hart, and Stewart [23].

Shneiderman [116] summarizes research results regarding display rates, response time, and user productivity. As he points out, the results indicate that productivity increases with decrease in response time. Error rates increase with response times that are too short or too long. In a system-guided dialogue, very short response times seem to pressure the novice user into actions. Very long response times disrupt the user's attention. Experienced users try to adapt to the pace of the system, often by tuning the command set for more efficient use.

Reisner [103] reports on an empirical study of query languages. In this study he defines a measure of learnability and ease-of-use. Based on the results he provides feedback for query language design. In addition, the report comments on methods for evaluating human factors and tries to show the reader how to interpret the results of such studies.

Embley and Nagy [35] review work on text editing and input/output techniques. Under controlled conditions, keystroke models, as provided in the work by Card, Moran, and Newell [25], can predict task time for expert users. More comprehensive models such as Card's GOMS model [24, 26] allow prediction of the user's choices in alternative routine editing tasks. Error detection and correction can account for up to 50% of the task time.

A variety of studies have compared different selection and positioning techniques based on different hardware devices [65, 25]. The results of those studies do not always agree. One case study [19] of several techniques for performing selection-positioning tasks examines the influence of lexical, pragmatic, and syntactic parameters of techniques on the ease of performing a particular task.

Perlman [94] examined the menu technique through an empirical study and derived some aids for low-level implementation decisions such as menu size, ordering of menu entries, and form of entry selection. Card, Pavel, and Farrell [27] examine windowing techniques, i.e., what features of windows are important for design of a user interface, and their benefits to users. The concept of a window working set (similar to the working set concept in operating systems) is introduced for addressing the problem of space constraints on window use. Gait [46] discusses the issue of window shapes. He observes that a collection of windows on a screen are aesthetically more pleasing if the dimensions of the windows adhere to the rule of a golden triangle, i.e., a triangle with Euclid's golden ratio.

In summary, there is considerable understanding of the basic perceptual requirements of displays. However, there is little systematic understanding of the interaction between display and the user's ability to perform a cognitive task.

5.3. Pictorial Representations

The form in which information is represented strongly affects the time it takes for the human mind to process it. It is commonly accepted that human visual orientation allows us to process visual information in different ways. As Raeder [98] points out, vision allows instant random access to any part of a picture, providing detailed and overall views; text, however, is scanned sequentially. Pictorial representation provides more dimensions according to which information can be laid out. Text is basically a one-dimensional stream of characters, augmented by formatting (such as paragraphs) and font changes (e.g., bold or italics). Pictures support three dimensions that can be augmented with properties such as shape, size, color, texture, direction, and distance. Because of the richer pictorial language, information can be encoded more compactly. The transfer rate of information by pictures is generally higher as the human eye is set up for real-time image processing. Pictures that incorporate real world objects can simplify understanding of abstract ideas. The pictorial representation allows us to refer to objects directly. In textual representation objects can only be referred to indirectly through names, so dependencies are less easily communicated. Pictures can represent the real world, thus making the representation more natural to the user. Animation of pictures, e.g., dials or plots, can reflect changes over time more easily than changing textual representations.

Pictures can be metamorphically enriched by making use of the additional dimensions of the pictorial language. As McCleary discusses, [76], size, value, direction, texture, shape, and color provide six variables according to which graphical symbols can be classified. Permutations and combinations of these variables provide for a large graphical vocabulary.

The appropriate symbolization of information, i.e., translation of textual and numerical information into graphical form, is a task in which graphic artists and illustrators have extensive experience. As Tufte shows [133], graphics can be an effective way to describe, explore, and summarize sets of data. However, graphics can easily be misused to produce what he calls *chart junk*. Tufte gives some guidelines for excellent statistical graphics. In his words, graphics can be more precise and revealing than statistical computation, and communicate complex ideas clearly, precisely, and efficiently.

5.4. Design Principles

The design principles summarized in this section have several sources: authors such as Newman and Sproull [88], Foley and van Dam [41], Good [50], and those summarized in Shneiderman [115]); a questionnaire-based study [33]; and the cognitiven psychology realm [26]. Despite the diversity of sources, there is considerable agreement on the principles. Some apparent contradictions among principles occur because the criteria summaries do not clearly differentiate between classes of users. Examples include request for redundancy in the command set, and system vs. user control over the dialogue. The design criteria are grouped as follows:

- Know the user — adapt to the user's capabilities; adapt wordiness of both input and output to the user's needs; allow choice of entry patterns; provide shortcuts for knowledgeable user; provide guidance when desired; use the user's model for the problem domain.
- Consider personal worth of the user — leave action initiation and control (e.g., abortion of command) to the user; allow the user to decide on dialogue technique and tailor the dialogue pattern; give quick and meaningful feedback on state and progress of interaction and execution; do not require the user to supply information that is already available; maintain a well-balanced social element in the dialogue.
- Reduce learning — provide consistency and clear wording; reduce modality; maintain uniformity and homogeneity; avoid overloading of commands; provide self-explanatory commands and help when requested or needed.
- Engineer for errors — provide consistency and uniformity; handle all possible input combinations; provide good error messages; eliminate common errors; provide

protection from costly errors; provide support for correction of errors; provide context-sensitive online assistance.

- Know the application — present a natural image of the application system; provide problem-oriented powerful commands; carry forward a representation of the user's knowledge base about the application's problem domain.

Card et al. [26] and Buxton [18] consider the time component (the expected performance requirements) to be an additional design criteria. Card et al. further propagate a set of principal steps that should be followed to achieve good user interface design:

1. Set goals to be met (performance requirements, target user population, problem and user-oriented definition of tasks),
2. Specify methods to accomplish the tasks,
3. Tune performance for experts by tailoring methods,
4. Define alternative methods such that alternatives are easily understood conceptually,
5. Design error recovery methods,
6. Analyze sensitivity of performance predictions to assumptions.

Early in the process of designing a system, the psychology of the user and the user interface should be considered.

These design principles and guidelines are based on experience and opinions. There is a need for more objective measures to determine the quality of user interface designs through evaluation. In addition, there is a need for formalized knowledge of design principles and a way to embed them into a system for building user interfaces (see section 6.2).

6. Implementation Considerations

This section discusses user interfaces from the view of the application implementor, who can be faced with quite a different interface from the user interface system. A user interface system may be easy to use, but hard to interface with an application program. The interface may provide limited functionality, functionality not appropriate for the application, or a certain interface style, which would not allow the implementor to specify his own style. The following examines how application user interfaces have been implemented.

In the simplest form, the application programmer uses I/O routines provided by the programming language or by a subroutine library. Often these facilities handle specific input and output devices or device classes (separate routines for terminal and file I/O) and different calling parameters for different terminals. This results in device-dependent applications that are difficult to port. Device-independent I/O models make applications more portable. Examples of such models include uniform file and device I/O in Unix, virtual device interfaces such as the Unix curses and termcap facility, and standard library routines for input processing such as the Unix library routines for text input handling. Increases in sophistication of user interface functionality results in more complex software. Since a considerable part of an application represents user interface code, and enhancements to software, there is a desire to localize the user interface code, to share and reuse the code across applications, and provide higher-level functionality by predefined user interface packages.

The term user interface management system (UIMS) has been coined [66, 132] for tools that assist in the development of good interactive application systems. The application developer expects benefits from a UIMS [18, 125, 58] such as:

- device independence of the application,
- reduction in programming effort,
- reuse for several applications resulting in more consistency, and
- quicker and more effective prototyping and tailoring of the user interface of an application.

The UIMS is expected to provide functionality at the appropriate level to model effectively the user's view of the application without imposing preconceived policies on the form of dialogue. Further, it is expected to be decoupled from the application to permit tailoring of the dialogue without affecting the application part, while still maintaining a correct and consistent display of the application and its state.

The following section discusses architectural models of user interface software, examines tools for developing user interfaces, and elaborates on the desire for portability and the necessary standardization efforts.

6.1. Architectural Models

Several models have been developed to define and understand the architecture of user interface systems and are summarized in this section. Each of the models addresses a different aspect of the user interface architecture. The first model presents a set of virtual machines that provide different levels of abstraction, reducing dependency on physical devices and the application. The second model captures three different control architectures. The third model tries to capture the structure of the dialogue, taking a language-oriented view. Proposals have been made to compensate for deficiencies in the latter model.

One model introduces levels of abstraction to address the problem of dependence on the physical device and of decoupling from the application. These levels of abstraction are the *logical device*, the *virtual terminal*, the *external view*, and the *dialogue socket* [29].

- The logical device level insures device independence. Two examples of logical device interfaces, the Unix curses and termcap facility [134] and the GKS standard interface [61], were discussed earlier in the document. Since logical devices are building blocks for higher-level logical devices and interaction techniques, they introduce a hierarchy [22]. Borufka, Kuhlmann, and ten Hagen [15] introduces the notion of dialogue cell as a method for defining interactions from logical devices based on GKS. Green [52] describes a catalog of graphical interaction techniques.
- The virtual terminal level introduces the notion of terminal sharing. It provides the notion of viewport, upon which higher-level abstractions can be implemented. Dialogue mechanisms such as menus use viewports as building blocks.
- The external view refers to the specification of an external, displayable representation for application defined abstractions, i.e, objects. Multiple views, tailored to the needs of the user's conception of the application's abstract model, can be provided. Object-oriented systems such as Smalltalk were forerunners of this idea. Prior to that, the notion of views was common practice in DBMS.
- The dialogue socket contributes to the dialogue-independence of an application. It provides a high-level interface between the application and the user interface, allowing for a decoupling of user interface and application development.

The interaction between the application and the user interface is described in a second model, the control architecture model. Three kinds of control architecture can be identified: *internal, external*, and *mixed* [125].

- In the internal control model, the application is in charge of the flow of control. By calling on functions in the user interface, it determines the flow of the dialogue.
- The external control model takes the inverse view, and the application is provided as a set of functional units, which the user interface system invokes in response to user interactions. Hence the user or user interface is in control of the dialogue sequence.

- The mixed control model supports concurrent execution of the application and the user interface in separate processes. They communicate through a message system. The user interface system must handle two asynchronous interfaces: the end user and the application. Such a model provides more flexibility in that it can deal with user or application-generated events that are not part of the main flow of dialogue.

The composition of an application dialogue from available building blocks can be formally specified. In a language-oriented view [42, 20], several aspects of the interaction dialogue have to be considered: pragmatics, lexical elements, syntax, and semantics. Different formalisms are most appropriate for each of these aspects. This architectural model has some shortcomings, in that it does not clearly address issues of two-dimensional and graphical display, and issues regarding the interaction between user input and feedback of the system. Shaw et al. [113] refer to the capturing of this interplay between input and output as *prosody*. They point out that attention must be paid to the way decisions are made about format and prosody. By localizing this information, policy decisions regarding the layout can be defined in a database, and style definitions can provide initial choices (an idea already present in the Scribe text formatting system [102]).

Shaw [114] presents a new model for input/output that overcomes the deficiencies of the classical language-oriented model. Her model suggests a software organization that supports handling of application-defined data abstractions, composition and maintenance of a display image, and processing of application and user events.

6.2. Development Tools

Tools for developing user interface software come in two categories: software building blocks and languages. Building blocks are made available in the form of toolboxes or in the form of generation systems. Languages exist in the form of programming languages for specification of the interface between the application and the user interface, and in the form of specification languages for user interface designers to define the appearance and behavior of the user interface. Each of these categories will be treated in turn in the next two sections.

6.2.1. Toolboxes and Generation Systems

Tanner and Buxton [125] discuss an evolving model of user interface management, i.e., the combination of the process of building a user interface and the runtime architecture of the user interface. They introduce the terms *module builders* and *glue systems* to refer to toolboxes and generation systems.

Toolboxes provide a library of building blocks from which a user interface can be manufactured. The user interface builder may be provided with tools to enhance the building block set, e.g., icon and font editor to add application specific symbols. The building blocks may provide different levels of functionality. The Macintosh Toolbox [5] and Microsoft Windows [70] are two examples of toolboxes from which the builder will pick building blocks and assemble them to form the preferred user interface. Certain aspects of the interface are predefined, e.g., the placement of a menu; others are in the control of the builder. In contrast, Andrew [110] is an example of a toolbox that provides a very high level interface and imposes built-in policies for the user interface appearance and behavior.

User interface generation systems allow the builder to formally specify the appearance and behavior of the user interface. This specification is then processed; the use of appropriate building blocks to achieve the effect is determined; and the information is deposited in a knowledge base. The runtime support of the system executes the user interface by interpreting this knowledge base and executing the library routines. Examples of such generation systems are products such as Apollo/Dialog [4], and research systems such as Tiger [66] and Cousin [58]. Some generation systems differ in that the user interface is not specified by a textual description, but through an interactive tool (e.g., Flair [141] and Menulay [21]). The interactive specification of the user interface and the direct interpretation of the knowledge base ease the construction and allow for rapid prototyping. Descriptions and comparisons of several user interface management systems can be found in Buxton et al. [21] and Olsen et al. [92].

It is interesting to note that syntax-directed editor systems have gone through a similar evolution. There are handcrafted systems [127], systems generated from formal descriptions [77], and systems tailored to rapid prototyping by providing interactive specification and direct interpretation of the same [39]. They can be considered user interface management systems [38] with limitations in that they usually do not provide functionality for arbitrary positioning and graphical output.

Issues to be addressed are the separation of policy and mechanism [113], the inclusion of design guidelines as a knowledge base into the user interface generation system, and the ability to include new technologies and techniques such as speech input [141] and color into user interface systems. The separation of mechanism and policy will shed some light on the appropriate parameterization (user tailoring) of the user interface. Embedding design guidelines into the generation system requires a formalization of these guidelines and principles. Based on these guidelines and appropriate input from the application designer, the system generates a user interface. In the area of business graphics, an attempt has been made to capture such information into the system [47]. Rather than the user selecting presentation mechanisms, such as piecharts, histograms, and choice of color, the user specifies to the system the effect to be achieved by the presentation; that is, whether a trend analysis is intended, or the data is to be compared in absolute or relative terms. Based on the answers and the type of data, the system automatically determines an appropriate presentation.

6.2.2. Languages

Language support falls into two categories: programming languages and specification languages. Programming languages have been extended to provide more than the classical support [114]. Examples of such work are PS-Algol [81], extension of Pascal [68], Screen Rigel [108], and Taxis [95]. Extensions include predefined types that are specific to the user interface and language constructs for carrying out user interface operations. The alternative to language extensions is the provision of a subroutine library. Work has also been done to define interfaces to graphics systems in a language-independent manner [120], and to automatically generate language-specific interfaces and the necessary support routines for message-oriented procedure calls [100, 4].

A range of formal specification languages have been employed in specifying the appearance and behavior of user interfaces. Some are applications of existing notations with extensions; others are special purpose languages. Their purpose is to capture the syntactic nature, the format, and the control flow. Green's thesis [53] surveys different graph-based, grammar-based, and algebraic specification techniques. Extended forms of the Backus-Naur Form [13, 91, 56] and variations [32, 77] have been used. State transition diagrams and augmented transition networks have been employed [93, 62, 138, 64]. Petrinets and other graphical specification techniques are attractive candidates. Chi [28] compares and evaluates four axiomatic approaches for formally specifying the semantics of user interfaces. Bass [11] discusses a theory of grouping and a system that supports formal user specification of form-based user interfaces according to this theory. Special purpose languages are tailored to the needs of the application area such as languages for description of office information environments [34], or languages for specifying pictures [135]. Languages have been designed that are based on state transitions, yet include other aspects such as layout as well [53, 113]. Finally, languages exist for defining data representations and views on them in databases [1], syntax editors [77], and language systems [87].

6.3. Portability and Standards

Compatibility of systems is a desirable attribute because it permits integration and porting with little or no cost. Since software costs are a major factor in system development, compatibility is important to user interface technology. Compatibility is achieved through common agreement to public interfaces. This can happen informally by a large population accepting someone's interface (defacto standards), or by an authority documenting, publishing, and promoting an interface standard. Organizations such as ISO, CCITT, NBS, ANSI, IEEE, ACM, and the DoD are active in promoting standards related to user interface technology. The remainder of this section discusses different dimensions of portability and highlights some of the official and defacto standards.

6.3.1. Portability

Portability can be viewed from many angles: portability of users and of software; or portability across different operating systems, devices, machines, and languages. Users portability is a prominent aspect in user interface technology. As the interaction style and the command language differ from application to application and system to system, users are constantly in the process of learning new ways of communication (tower of Babel). This process can be made easier if uniformity and consistency exist. Through reuse, user interface management systems can contribute by providing a consistent set of vocabulary and gestures. This, however, conflicts with the user interface designer's attempt to present a conceptual view of the application that is close to the user's mental model of the application and makes the interaction more natural and learnable.

Portability of languages has two aspects. One aspect is support for an interface between the user interface system and the application such that applications can be written in different programming languages. Multiple language support can be provided in several ways. One approach is taken by VAX/VMS and Apollo/DOMAIN where all languages share a runtime environment, and interlanguage calls are supported. Limitations, however, may exist in some languages in the support of data abstraction and event processing. Another approach makes use of the property that the application and the user interface run in separate processes. The message protocol or remote procedure call mechanism, if defined appropriately, can interconnect processes running different language systems [101]. A second aspect of language portability is support for multiple natural languages. As shown in the Macintosh product, all dialogue text can be kept in a database separate from the application code. However, the situation has to be considered for several people with different natural languages using the same machine.

User interfaces and applications are usually dependent on the operating system. For example, the Macintosh Toolkit and Microsoft's Windows were designed for and implemented on single process operating systems. Design decisions inhibit those user interface systems from being easily made available on other operating systems. Multiple process architectures for user interfaces make use of message systems and remote procedure call mechanisms.

Portability across input and output devices has been handled by defining virtual interfaces. Device-specific knowledge is hidden in the implementation of the interface. Examples are the GKS logical devices for graphical/bitmap displays, and the Unix file I/O interface and curses/termcap package for character terminals. By building on such virtual interfaces, large portions of the user interface system can be kept device independent. This general statement is true for devices within a device class. However, the functionality of devices in one class may not be sufficient to simulate the functionality of devices in another class. Therefore, the user interface design may have to be adapted for different device classes. With an appropriate user interface management system, this information can be localized in the user interface specification. The application does not have to be affected.

Porting of applications to other machines includes adaptation of the user interface. This can be accomplished in two ways: by keeping the user interface system portable; or by defining a standard interface between the application and the user interface that is implemented by different user interface systems on different machines. In a network environment, a third alternative is possible. Access to an application from another machine can be provided without a port of both user interface and application. In the mixed control model, the user interface and the application are partitioned into separate processes. Network-wide interprocess communication or remote procedure call service makes a distributed setup possible. Other partitions, such as at a virtual device level, can also be envisioned. The network bandwidth will determine the effective feasibility of different distribution scenarios.

An interesting special case of porting application between devices or user interface systems is the interconnection of applications, i.e., driving one application from another. Most interactive applications cannot be executed from another program, whether user program or batch processing program, due to restrictions in the operating and runtime system. Even on operating systems that encourage coupling of applications, such as UNIX, this is not always possible because the application has embedded knowledge about conversing with an interactive device.

6.3.2. Official Standards

Official standards are most prevalent at the device level. Only a few efforts exist at more abstract levels of the user interface.

Character sets are encoded in ANSI's ASCII standard or in defacto standards such as IBM's EBCDIC. The ASCII standard exists for 7-bit and 8-bit encoding as well as for a multilingual graphic character set.

ANSI has defined a standard (X3.64) for the interface between CRT terminals and computers. This standard is very close to the functionality of the DEC VT100 terminal. Other terminal hardware vendors have adopted this standard by providing an emulation mode. (As it turns out, it is the VT100 terminal more often than the X3.64 standard that is being emulated.) On workstations with bitmap displays, window systems provide one window type to emulate this terminal standard [97]. Such an emulation permits the workstation to be treated as a terminal to another machine and allows programs to be ported and run on the workstation with little effort, but without adaptation to additional capabilities of the window system.

Several standards exist in the graphics area. Close to the hardware level, a standard for a device-independent interface to different graphics equipment was defined in the form of a *virtual device interface/metafile* (VDI/VDM), now known as computer graphics interface/metafile under ANSI X3H3.3. The metafile specification defines a mechanism for storing and transmitting graphical images. Below the VDI level, NAPLPS defines a standard method for encoding character and point information for driving videotext devices. The GKS defines a virtual graphics model and provides an interface to the application. A device-independent implementation of GKS resides on top of the VDI level. Bono [14] explains the interrelationship of these and other graphic standards in more detail and discusses their role in information interchange.

GKS was adopted as a standard by ISO/TC 97 (ISO/DIS 7942). The standard includes proposals for language bindings of the GKS package to common languages such as Ada (ISO/DP 8651/3). Proposals exist for a GKS for three dimensions (ISO/DP 8805), and a metafile description for graphical representations (ISO/DP 8632). Waggoner [136] points out differences in the two standards and gives reasons for the success of GKS. Developing UIMS on top of GKS has proven feasible, but has also uncovered some shortcomings due to the mismatch between the model of the graphic system and the UIMS [132]. The tools available in the standard graphic system are inadequate for supporting static or dynamic division of the screen into windows [106] and for providing a linkage between input and output [92].

Standardization efforts are in progress for document interchange formats. ISO has proposals for standardization of text preparation and interchange for text structures (ISO/DP 8613/1-6) and for processing and mark-up languages (ISO/DP 8879). For a status report on this and other efforts in document interchange, the reader is referred to Horak [59].

Military guidelines and advisories exist in various forms such as MIL-STD-1472C (Human Engineering Design Criteria for Military Systems, Equipment, and Facilities) [31] and ESD-TR-83-122 (Design Guidelines for the User Interface to Computer-Based Information Systems) [118].

6.3.3. Defacto Standards

UNIX, which has two major branches of evolution (Berkeley 4.2 and AT&T System V), is becoming a defacto industry standard for operating systems. UNIX has made several advances regarding user interfaces. It provides a command language (*sh* and its variants *csh* and *ksh*) that enables programmability and easy extensibility of the command set. It supports command history, execution of previous commands, and editing of commands. Through the termcap interface, Unix provides terminal-independent access to CRT terminals. The uniform treatment of terminal, file, and special device I/O allows simple execution of interactive programs from other programs, including a facility from the shell to specify an input stream to an interactive program invoked by it.

Several commonly accepted information interchange formats are in common use. Xerox, as part of its office information system effort, has defined the InterScript format [6] as a document representation format. PostScript [2] is a device-independent page description language that is quickly becoming the industry standard for printing high-quality integrated text and graphics. PostScript has been incorporated into laser printers manufactured by Apple Computer, Allied Linotype, QMS Incorporated, and Dataproducts Corporation. It is supported by document production systems such as Scribe, MacWrite/MacPaint, etc. Microsoft is propagating SYLK, a symbolic link file format for ASCII representation of data for transfer and exchange between different applications such as the Multiplan spreadsheet program [82].

7. Summary of Issues

The intent of this section is to summarize issues regarding different aspects of user interfaces. The resolution of the summarized issues is viewed as advancing the state-of-practice of user interface technology and increasing the use of advanced user interface technology in the work environment of software engineering. The issues fall into two groups. The first group deals with the fact that one cannot expect to find homogeneous computing environments, but must expect to cope with heterogeneous computer equipment. The second group focuses on the heterogeneity of applications (software tools) and users, and the ability of the user interface to appropriately support it.

7.1. Heterogeneous Computing Environments

Today there is a proliferation of workstations. These workstations can be placed in one environment by wiring them together into one local area network. Furthermore, users may work from home using a modem and a character terminal or workstation. This physical connection, however, does not make the resources available to the user in a transparent manner. applications are bound to specific hardware and cannot be executed remotely on different hardware. The display functionality provided on one workstation may not be available on another one. The connection bandwidth between different workstations may make certain modes of operation intolerable. Issues of system partitioning, portability, and distribution need to be addressed in order to bring improvements to the work environment of software engineers, most likely heterogeneous computing environments.

Many tools, both interactive and noninteractive, make up the work environment of software engineers. Software development environment builders cannot be expected to provide all tools themselves. Tools are being developed by different groups on different hardware and software, providing different user interfaces. Integrating this multiplicity of tools into a consistent and uniform environment is a challenging task. Proliferation of hardware, at least on the desk of a single software engineer, must be reduced. Tools must be able to interact with each other, share and exchange information. Furthermore, it is desirable for a user to interact with one conceptually uniform user interface, even for tools on different hardware and operating systems. Existing tools cannot always be discarded and must be adapted to the user interface of the environment. The increase in sophistication of user interface technology and the lack of interface standardization at a high functional level has made the resolution of these problems a difficult task.

7.2. Heterogeneous User Groups

Users are a heterogeneous population. For users to be most effective, the user interface of a system should adapt to the needs and capabilities of a user. Preconceived notions of expertise in a system are often more of a hindrance than a help because the degree of expertise can vary greatly even within one application and can change over time by nonuse of the system. Given appropriate tools, the user can perform some tailoring of the user interface himself.

User interface management systems have simplified the process of building and tailoring user interfaces to applications. However, they often reflect their designer's view of user interactions. Instead, UIMSs should provide a set of mechanisms to provide user interaction services and a facility for a user interface designer to indicate his choices regarding appearance and behavior of the user interface. The designer should be guided by known design principles and guidelines that, to date, have been expressed informally. This knowledge should be formalized and embedded in the UIMS.

There is a tension between the desire to tailor to individual users and tools and the attempt to achieve the appearance of an integrated software development environment with a uniform user interface. At issue is the appropriate balance between flexibility and standardization of the user interface in a software development environment. Certain flexibility should be provided by and be built into the user interface, e.g., support for different degrees of expertise. Certain properties of the user interface should be specified once for a particular environment and left relatively stable. If parameterization to individuals is supported, there should be a simple way for changing it to that of a different individual in short notice, e.g., one person walking up to another person's display and being able to help him without being confronted with tailored keybindings.

The techniques for evaluation of user interfaces require further attention. Work derived from applied cognitive psychology resulted in some models for quantitatively measuring basic interaction steps. Additional models that address other aspects of user interfaces need to be developed together with tools for carrying out the evaluation. Benchmarks must evolve to allow for a fair comparison of results.

Acknowledgements

As a starting point for this report, the author used a bibliography on interactive display interfaces from the Descartes Project at the Computer Science Department, Carnegie-Mellon University, with Mary Shaw as principal investigator. This bibliography contributed approximately 30% of the references in this report.

The author is indebted to colleagues at the Software Engineering Institute and the Computer Science Department for their discussions. He is especially grateful to Mario Barbacci, Joe Newcomer, John Nestor, Mary Shaw, and Nelson Weiderman for providing comments on earlier drafts of this report, and to Purvis Jackson, Suzanna Garreau, and Bill McSteen for editing and document production.

Appendix A: Trademarks

Ada is a registered trademark of the U.S. Government, Ada Joint Program Office.

Apollo and DOMAIN are registered trademarks of Apollo Computer, Inc.

DEC, MicroVAX, VAX, VMS, and VT are trademarks of Digital Equipment Corporation.

Frogger is a trademark of Sega and Sierra On-line, Inc.

Knowledge Craft is a trademark of Carnegie Group, Inc.

Knowledge Engineering Environment is a trademark of Intellicorp.

Jazz is a trademark of Lotus Development Corporation.

MacProject, MacTerminal, and Apple are trademarks and Macintosh is a trademark licensed to Apple Computer, Inc.

POSTSCRIPT is a trademark of Adobe Systems Incorporated.

Scribe is a registered trademark of UNILOGIC, Ltd.

Smalltalk-80 and Xerox Star are trademarks of Xerox Corporation.

Sun is a trademark of Sun Microsystems, Inc.

UNIX is a registered trademark of Bell Laboratories.

References

[1] E. H. Sibley (editor).
Special Issue: Data-Base Management Systems.
ACM Computing Surveys, 1976.

[2] Adobe Systems.
PostScript Language Reference Manual.
Addison-Wesley Publishing Company, Reading, MA, 1985.

[3] Apollo Computer Inc.
DOMAIN User Manual
Chelmsford, MA, 1985.

[4] Apollo Computer Inc.
DOMAIN/Dialog User's Guide
Chelmsford, MA, 1985.

[5] Apple Computer.
Inside Macintosh.
Apple Computer, San Jose, CA, 1985.

[6] Robert M. Ayers and James L. Horning.
Interscript: A Proposal for a Standard for the Interchange of Editable Documents.
1984.

[7] R. Baecker, W. Buxton, and W. Reeves.
Towards Facilitating Graphical Interaction: Some Examples from Computer-Aided Musical Composition.
In *Proc. 6th Man Computer Communications Conference*, pages 197-207. Computer Systems Research Group, Ottawa, Canada, May, 1979.

[8] J. Eugene Ball.
Alto as Terminal.
1980.
Documentation of an Alto program.

[9] J. Eugene Ball.
Canvas: the Spice graphics package.
Technical Report Spice Document S108, Carnegie-Mellon University, Dept. of Computer Science, August, 1981.

[10] David R. Barstow, Howard E. Shrobe, and Erik Sandewall, eds.
Interactive Programming Environments.
McGraw-Hill, 1984.

[11] Leonard J. Bass.
An Approach to User Specification of Interactive Display Interfaces.
IEEE Transactions on Software Engineering SE-11(8):686-698, August, 1985.

[12] C. G. Bell, A. Newell, and D. P. Siewiorek.
Computer Structures: Principles and Examples.
McGraw Hill, Computer Systems Series, 1982.

[13] T. Bleser and James D. Foley.
Towards Specifying and Evaluating the Human Factors of User-Computer Interfaces.
In *Human Factors in Computer Systems Proceedings*. March, 1982.

[14] Peter R. Bono.
A Survey of Graphics Standards and Their Role in Information Interchange.
IEEE Computer , October, 1985.

[15] H.G. Borufka, H.W. Kuhlmann, and P.J.W. ten Hagen.
Dialogue Cells: A Method for Defining Interactions.
IEEE Computer Graphics and Applications 2(5):25-33, July, 1982.

[16] G. P. Brown, R. T. Carling, C. F. Herot, D. A. Kramlich, and P. Souza.
Program Visualization: Graphical Support for Software Development.
Computer 18(8):27-35, August, 1985.

[17] J. L. Burch.
Computers: The Non-Technical (Human) Factors.
The Report Store, 1984.

[18] William Buxton and Richard Sniderman.
Iteration in the Design of the Human-Computer Interface.
In Paul Stager (editor), *Quality of Work Life and Human Factors*, pages 72-81. Human Factors Association of Canada, September, 1980.
13th Annual Meeting.

[19] William Buxton.
An Informal Study of Selection Positioning Tasks.
Graphics Interface :323-328, September, 1982.

[20] William Buxton.
Lexical and Pragmatic Considerations of Input Structures.
Computer Graphics 17(1):31-37, January, 1983.

[21] William Buxton, M. R. Lamb, D. Sherman, and K. C. Smith.
Towards a Comprehensive User Interface Management System.
Computer Graphics 17(3), July, 1983.

[22] D. U. Cahn and A. C. Yen.
A Device Independent Network Graphic System.
Computer Graphics 17(3):167-173, July, 1983.

[23] A. Cakir, D.J. Hart, and T.F.M. Stewart.
Visual Display Terminals.
John Wiley & Sons, 1980.

[24] Stuart K. Card.
Studies in the Psychology of Computer Text Editing Systems.
Technical Report SSL-78-1, Xerox PARC, August, 1978.

[25] Stuart K. Card, Thomas P. Moran, and Allen Newell.
The Keystroke-Level Model for User Performance Time with Interactive Systems.
Communications of the ACM 23(7):396-409, July, 1979.

[26] Stuart K. Card, Thomas P. Moran, and Allen Newell.
The Psychology of Human-Computer Interaction.
Lawrence Erlbaum Associates, Hillsdale, N.J., 1983.

[27] Stuart K. Card, M. Pavel, and J. E. Farrell.
Window-Based Computer Dialogues.
In *Interact '84*, pages 355-359. IFIP, Elsciver, Science Publisher, September, 1984.

[28] Uli H. Chi.
Formal Specification of User Interfaces: A Comparison and Evaluation of Four Axiomatic Approaches.
IEEE Transactions on Software Engineering SE-11(8):671-685, August, 1985.

[29] Joelle Coutaz.
Abstractions for User Interface Design.
Computer 14(3):21-34, September, 1985.

[30] Nastec Corp.
DesignAid, a graphics editor.
Product Description.
1985

[31] US Dept. of Defense.
Human Engineering Design Criteria for Military Systems, Equipment, and Facilities.
Technical Report MIL-STD-1472C, DoD, May, 1981.

[32] Veronique Donzeau-Gouge, Gerard Huet, Gilles Kahn, and Bernard Lang.
Programming Environments Based on Structured Editors: The MENTOR Experience.
In David R. Barstow, Howard E. Shrobe, and Erik Sandewall (editors), *Interactive Programming Environments*. McGraw-Hill, 1984.

[33] W. Dzida, S. Herda, and W. D. Itzfeldt.
User-Perceived Quality of Interactive Systems.
IEEE Trans. on Software Engineering SE-4(4):270-278, July, 1978.

[34] C. A. Ellis and G. J. Nutt.
Office Information Systems and Computer Science.
Computing Surveys 12(1):27-60, March, 1980.

[35] David W. Embley and George Nagy.
Behavioral Aspects of Text Editors.
ACM Computing Surveys 13(1):33-70, March, 1981.

[36] Kenneth B. Evans, Peter P. Tanner, and Marceli Wein.
Tablet-Based Valuators that Provide One, Two, or Three Degrees of Freedom.
In Henry Fuchs (editor), *Computer Graphics*, pages 91-97. ACM, August, 1981.

[37] Peter H. Feiler and Raul Medina-Mora.
An Incremental Programming Environment.
IEEE Transactions on Software Engineering SE-7(5), September, 1981.

[38] Peter H. Feiler and Gail E. Kaiser.
Display-Oriented Structure Manipulation in a Multi-Purpose System.
In *Proceedings of the IEEE Computer Society's Seventh International Computer Software and Applications Conference (COMPSAC '83)*. November, 1983.

[39] Peter H. Feiler, Fahimeh Jalili, and Johann H. Schlichter.
An Interactive Prototyping Environment for Language Design.
In *Proceedings of Hawaii International Conference on System Sciences*. ACM,IEEE, January, 1986.

[40] C.N. Fischer, A. Pal, D.L. Stock, G.F. Johnson, and J. Mauney.
The POE Language-Based Editor Project.
In *Software Engineering Symposium on Practical Software Development Environments*. ACM SIGSOFT and SIGPLAN, May, 1984.

[41] James.D. Foley and Andries van Dam.
Fundamentals of Interactive Computer Graphics.
Addison-Wesley, 1984.
Revised edition; originally issued 1982.

[42] James D. Foley, Victor L. Wallace, and Peggy Chan.
The Human Factors of Computer Graphics Interaction Techniques.
IEEE Computer Graphics and Applications 4(11):13-48, November, 1984.

[43] Mark S. Fox and Andrew J. Palay.
The BROWSE System: An Introduction.
In R. N. Oddy, S. E. Robertson, C. J. van Rijsbergen, and P. W. Williams (editor), *Information Choices and Policies*, pages 183-190. ASIS, Butterworth's, London, 1979.

[44] Mark S. Fox and Andrew J. Palay.
Browsing through databases.
Information Retrieval Research.
Butterworth's, London, 1981, pages 310-324.

[45] R.Furuta, J. Scofield, and A. Shaw.
Document Formatting Systems: Survey, Concepts, and Issues.
ACM Computing Surveys 14(3):417-472, September , 1982.

[46] Jason Gait.
An Aspect of Aesthetics in Human-Computer Communications: Pretty Windows.
IEEE Transactions on Software Engineering SE-11(8):714-717, August, 1985.

[47] S. Gnanamgari.
Information Presentation Through Automatic Graphic Displays.
In *Proceedings of the International Conference*, pages 433-445. Online Conferences Limited, Online Publications Ltd., Middlesex, UK, October, 1981.

[48] Adele Goldberg.
The Influence of an Object-Oriented Language on the Programming Environment.
Interactive Programming Environments.
McGraw Hill, 1984, pages 141-174, Chapter 8.

[49] Adele Goldberg.
Smalltalk-80: The Interactive Programming Environment.
Addison-Wesley, 1984.

[50] Michael Good.
Etude and the Folklore of User Interface Design.
In *Proceedings of the ACM SIGPLAN/SIGOA Symposium on Text Manipulation*, pages 1-26. June, 1981.
Also Office Automation Group Memo OAM-030, Massachusetts Institute of Technology, March 1981.

[51] Robert B. Grafton.
Visual Programming.
Computer 18(8):6-9, August, 1985.

[52] Mark W. Green.
A Catalogue of Graphical Interaction Techniques.
Computer Graphics 17(1):46-52, January, 1983.

[53] Mark W. Green.
The Design of Graphical User Interfaces.
PhD thesis, University of Toronto, April, 1985.

[54] A. N. Habermann and D. Notkin.
The GANDALF Software Development Environment.
In *Proceedings of the Second International Symposium on Computation and Information*. Monterrey, Mexico, September, 1983.

[55] M. Hammer, R. Ilson, T. Anderson, E. J. Gilbert, M. Good, B. Niamir, L. Rosenstein, and S. Schoichet.
The implementation of Etude, an integrated and interactive document production system.
In SIGPLAN Notices (editor), *Proc. ACM SIGPLAN SIGOA Symp. Text Manipulation*, pages 137-141. June , 1981.

[56] Wilfred J. Hansen.
User Engineering Principles for Interactive Systems.
In *AFIPS Conference Proceedings*, pages 523-532. American Federation of Information Processing Society, Las Vegas, November, 1971.
Also published in *Interactive Programming Environments* [10].

[57] P.J. Hayes.
Cooperative Command Interaction Through the COUSIN System.
In *Proceedings of the International Conference on Man/Machine System*. University of Manchester Institute of Science and Technology, Institution of Electrical Engineers, London, July, 1982.

[58] P. J. Hayes, P. A. Szekely, and R. A. Lerner.
Design Alternatives for User Interface Management Systems Based on Experience with Cousin.
In *Proceedings of CHI85*. San Francisco, April, 1985.

[59] W. Horak.
Office Document Architecture and Office Document Interchange Formats: Current Status of International Standardization.
IEEE Computer , October, 1985.

[60] Bruce MacKinlay.
The Interleaf Publishing System.
Unix World 1(5):94-97, December, 1984.

[61] International Standards Organization.
Graphical Kernel System.
ISO/DIS 7942 REV.
1985

[62] Robert J. K. Jacob.
A State Transition Diagram Language for Visual Programming.
Computer 18(8):51-59, August, 1985.

[63] A. Janda.
Index to Post-1980 Literature on Computer-Human Interaction.
ACM SIGCHI Bulletin , July, 1983.

[64] Abid Kamran and Michael B. Feldman.
Graphics Programming Independent of Interaction Techniques and Styles.
Computer Graphics 17(1):58-66, January, 1983.

[65] John Karat, James E. McDonald and Matt Anderson.
A Comparison of Selection Techniques: Touch Panel, Mouse and Keyboard.
In *Interact '84*, pages 149-153. IFIP, Elsciver, Science Publisher, September, 1984.

[66] David J. Kasik.
A User Interface Management System.
ACM Computer Graphics 16(3):99-106, July, 1982.

[67] B.W. Kernighan.
PIC - A Language for Typesetting Graphics.
ACM SIGPLAN Notices 16(6):92-98, June, 1981.

[68] J.M. Lafuente and D. Gries.
Language Facilities for Programming User-Computer Dialogues.
IBM J. Res. Develop. 22(2):145-158, March, 1978.

[69] David B. Leblang and Robert P. Chase, Jr.
Computer-Aided Software Engineering in a Distributed Workstation Environment.
In Peter Henderson (editor), *Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, pages 104-112. ACM, May, 1984.

[70] Phil Lemmons.
Microsoft Windows.
Byte Magazine :48-54, December, 1983.

[71] Ralph L. London and R. A. Duisberg.
Animating Programs Using Smalltalk.
Computer 18(8):61-71, August, 1985.

[72] Apple Computers.
Macintosh MacPaint.
Product Manual.
1984

[73] Apple Computers.
Macintosh MacTerminal.
Product Manual.
1984

[74] Apple Computers.
Macintosh MacWrite.
Product Manual.
1984

[75] James Martin.
Design of Man-Computer Dialogues.
Prentice Hall, Englewood Cliffs, NJ, 1973.

[76] George F. McCleary, Jr.
An Effective Graphic Vocabulary.
IEEE Computer Graphics & Applications 3(2):46-53, March/April, 1983.

[77] Raul Medina-Mora.
Syntax-Directed Editing: Towards Integrated Programming Environments.
PhD thesis, Carnegie-Mellon University, 1982.

[78] B. Melamed and R. J. T. Morris.
Visual Simulation: The Performance Analysis Workstation.
Computer 18(8):87-94, August, 1985.

[79] Norman Meyrowitz and Andries van Dam.
Interactive Editing Systems, Parts I and II.
Computing Surveys 14(3):321-415, September, 1982.

[80] Thomas P. Moran.
An Applied Psychology of the User.
ACM Computing Surveys 13(1):1-12, March, 1981.

[81] R. Morrison, A. L. Brown, A. Dearle, and M. P. Atkinson.
An Integrated Graphics Programming Environment.
Technical Report PPR-14-85, University of Glasgow, July, 1985.

[82] Microsoft Corp.
Microsoft Multiplan.
Product description and manual.
1984

[83] Allen Munro.
Filevision: The Data Base Worth a Thousand Words.
St. Mac :45-50, August, 1984.

[84] Brad A. Myers.
Displaying Data Structures for Interactive Debugging.
Master's thesis, MIT, June, 1980.

[85] Brad A. Myers.
The User Interface for Sapphire.
IEEE Computer Graphics and Applications 4(12):13-23, December, 1984.

[86] I. Nassi and B. Shneiderman.
Flowchart Techniques for Structured Programming.
SigPlan Notices SE-6(1), August, 1973.

[87] John R. Nestor, William A. Wulf, and David A. Lamb.
IDL - Interface Description Language.
Formal Description, CMU Department of Computer Science, 1982.

[88] William M. Newman and Robert F. Sproull.
Principles of Interactive Computer Graphics.
McGraw-Hill, 1979.

[89] Donald A. Norman.
Four Stages of User Activities.
In *Interact '84*, pages 81-85. IFIP, Elsciver, Science Publisher, September, 1984.

[90] D. Notkin.
Interactive Structure-Oriented Computing.
PhD thesis, Department of Computer Science, Carnegie-Mellon University, February, 1984.

[91] Dan R. Olsen Jr.
Automatic Generation of Interactive Systems.
Computer Graphics 17(1):53-57, January, 1983.

[92] Dan R. Olsen, W. Buxton, R. Ehrich, D. J. Kasik, J. R. Rhyne, and J. Sibert.
A Context for User Interface Management.
IEEE Computer Graphics and Applications 4(12):33-42, December, 1984.

[93] David L. Parnas.
On the Use of Transition Diagrams in the Design of a User Interface for an Interactive Computer System.
In *ACM Proceedings of the 24th National Computer Conference*, pages 379-385. July, 1969.

[94] Gary Perlman.
Making the Right Choices With Menus.
In *Interact '84*, pages 291-295. IFIP, Elsciver, Science Publisher, September, 1984.

[95] Michel Pilote.
A Programming Language Framework for Designing User Interfaces.
Proc. Sigplan '83 Symposium on Programming Languages Issues in Software Systems 18(6):118-136, June, 1983.

[96] A. Poggio, J. J. Garcia Luna Aceves, E. J. Craighill, D. Moran, L. Aguilar, D. Worthington, and J. Hight.
CCWS: A Computer-Based, Multimedia Information System.
IEEE Computer 18(10):92-103, October, 1985.

[97] Vaughan R. Pratt.
Standards and Performance Issues in the Workstation Market.
IEEE Computer Graphics and Applications 4(4):71-76, April, 1984.

[98] Georg Raeder.
A Survey of Current Graphical Programming Techniques.
Computer 18(8):11-25, August, 1985.

[99] H.R. Ramsey, and M.E. Atwood.
Human Factors in Computer Systems: A Review of Literature.
Science Applications Inc., Englewood, CO (available from NTIS as AD A075679).
1979

[100] R.F. Rashid, M. Jones, and R. Thompson.
Matchmaker: An Interface Specification Language for Distributed Processing.
In *Principles of Programming Languages*. ACM, January, 1981.

[101] R.F. Rashid and G.G. Robertson.
Accent: A communication oriented network operating system kernel.
In *Eighth Symposium on Operating System Principles*. ACM, November, 1981.

[102] Brian K. Reid.
Scribe: A Document Specification Language and its Compiler.
PhD thesis, Carnegie-Mellon University, October, 1980.

[103] Phyllis Reisner.
Human Factors Studies of Database Query Languages: A Survey and Assessment.
ACM Computing Surveys 13(1):13-32, March, 1981.

[104] Steven P. Reiss.
Graphical Program Development with PECAN Program Development Systems.
In *Proceedings of the SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*. April, 1984.

[105] Mary Riley and Claire O'Malley.
A Framework for Analyzing User-Computer Interactions.
In *Interact '84*, pages 86-91. IFIP, Elsciver, Science Publisher, September, 1984.

[106] David S.H. Rosenthal.
Managing Graphical Resources.
Computer Graphics 17(1):38-45, January, 1983.

[107] D. T. Ross.
Applications and Extensions of SADT.
IEEE Computer , April, 1985.

[108] L.A. Rowe and K.A. Shoens.
Programming Language Constructs for Screen Definition.
IEEE Trans. on Software Engineering SE-9(1):31-39, January, 1983.

[109] E. Sandewall.
Programming in the Interactive Environment: The InterLisp Experience.
ACM Computing Surveys 10(1), March, 1978.
Also published in *Interactive Programming Environments* [10].

[110] M. Satyanarayanan.
The ITC Project: A Large-Scale Experiment in Distributed Personal Computing.
In *Proceedings of the Networks 84 Conference, Indian Institute of Technology (Madras) October 1984*. North-Holland, 1984.
Also available as ITC tech report CMU-ITC-035.

[111] N. M. Delisle, D. E. Menicosy, and Mayer D. Schwartz.
MagPie -- An Interactive Environment for Pascal.
Technical Report CR-83-18, Tektronix, Inc., February, 1984.

[112] J. Seybold.
Xerox's 'Star'.
In *The Seybold Report*. Seybold Publications, Media, Pennsylvania, 1981.

[113] Mary Shaw, Ellen Borison, Michael Horowitz, Tom Lane, David Nichols, and Randy Pausch.
Descartes: A Programming-Language Approach to Interactive Display Interfaces.
Proc. Sigplan '83 Symposium on Programming Language Issues in Software Systems 18(6):100-111, June, 1983.

[114] Mary Shaw.
An Input-Output Model for Interactive Systems.
In *Proceedings of the NGI/SION Annual Symposium*. Utrecht, Netherlands, April, 1985.

[115] Ben Shneiderman.
Software Psychology: Human Factors in Computer and Information Systems.
Winthrop Publishers, Inc., Cambridge, Massachusetts, 1980.

[116] Ben Shneiderman.
Response Time and Display Rate in Human Performance with Computers.
ACM Computing Surveys 16(3):265-285, September, 1984.

[117] G. E. Sievert and T. A. Mizell.
Specification-Based Software Engineering with TAGS.
IEEE Computer , April, 1985.

[118] S. L. Smith and A. F. Aucella.
Design Guidelines For The User Interface To Computer-based Information Systems.
Technical Report SEI: ESD-TR-83-122, DTIC: AD-A 127, NTIS: 345/7, Electronic Systems Division, Hanscom Air Force Base, 1983.

[119] Spice Project.
Spice System Documentation.
Spice document series, Department of Computer Science, Carnegie-Mellon University.
1984

[120] Robert F.Sproull.
Omnigraph: Simple Terminal-Independent Graphics Software.
Technical Report CSL 73-4, Xerox Palo Alto Research Center, December, 1977.

[121] R.M. Stallman.
EMACS: The extensible, customizable self-documenting display editor.
In *ACM SIGPLAN/SIGOA Conference Text Manipulation*. SIGPLAN and SIGOA, June, 1981.
Extended version is published in *Interactive Programming Environments* [10].

[122] J. F. Stay.
HIPO and Integrated Program Design.
IBM Systems Journal 15(2), July, 1976.

[123] D. C. Swinehart.
Copilot: A Multiple Process Approach to Interactive Programming Systems.
PhD thesis, Stanford University, July, 1974.

[124] D. C. Swinehart, L. C. Stewart, and S. M. Ornstein.
Adding Voice to an Office Computer Network.
Technical Report CSL-83-8, Xerox Corporation, February, 1984.

[125] Peter P. Tanner and William A.S. Buxton.
Some Issues in Future User Interface Management System (UIMS) Development.
In *Workshop on User Interface Management*. IFIP, February, 1984.

[126] D. Teichroew and E. A. Hershey III.
PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems.
IEEE Trans. on SE SE-6(1), January, 1977.

[127] Tim Teitelbaum and Thomas Reps.
The Cornell Program Synthesizer: A Syntax-Directed Programming Environment.
Communications of the ACM 24(9), September, 1981.
Also published in *Interactive Programming Environments* [10].

[128] Warren Teitelman.
A Display-Oriented Programmer's Assistant.
In *Proceedings 5th International Joint Conference on Artificial Intelligence*, pages 905-915. 1977.
Also published in *Interactive Programming Environments* [10].

[129] Warren Teitelman and Larry Masinter.
The Interlisp Programming Environment.
IEEE Computer , April 1981.
Also published in *Interactive Programming Environments* [10].

[130] Warren Teitelman.
A Tour Through Cedar.
IEEE Software 1(2):44-73, April, 1984.

[131] L. Tesler.
The Smalltalk Environment.
Byte Magazine , August, 1981.

[132] James J. Thomas.
Graphical Input Interaction Technique (GIT).
Computer Graphics 17(1):5-30, January, 1983.

[133] Edward R. Tufte.
The Visual Display of Quantitative Information.
Graphics Press, Cheshire, CT, 1983.

[134] Bell Labs.
Unix System User's and Programmer's Manuals.
Manuals (ucb4.2 Unix manuals from Berkeley University).
1984

[135] C.J. Van Wyk.
A High-level Language for Specifying Pictures.
ACM Transactions on Graphics 1(2):163-182, April, 1982.

[136] Clinton N. Waggoner.
GKS, a Standard for Software OEMs.
Eurograph '83 :363-374, 1983.

[137] Janet H. Walker and Richard M. Stallman.
Systems That Grow With You: An Experiment With Supportive Software.
1982.

[138] Anthony I. Wasserman.
Extending State Transition Diagrams for the Specification of Human-Computer Interaction.
IEEE Transactions on Software Engineering SE-11(8):699-713, August, 1985.

[139] G. Williams.
The Apple Macintosh Computer.
Byte , February, 1984.

[140] Ian H. Witten and Bob Bramwell.
A System for Interactive Viewing of Structured Documents.
Communications ACM 28(3):280-288, March, 1985.

[141] Peter C.S. Wong and Eric R. Reid.
Flair -- User Interface Dialogue Design Tool.
ACM Computer Graphics 16(3):87-98, July, 1982.

Table of Contents