Human–Computer Interaction

Mike Sharples

I. INTERACTING WITH COMPUTERS

Human–computer interaction (HCI) is the study of the ways people interact with and through computers. It grew out of work on human factors (the U.S. term) or ergonomics (the European term) with the intellectual aim of analyzing tasks that people perform with computers and the practical concerns of designing more usable and reliable computer systems. As computers have infiltrated homes and businesses, the scope of HCI has broadened to include the cognitive, social, and organizational aspects of computer use. HCI can provide techniques to model people's interactions with computers, guidelines for software design, methods to compare the usability of computer systems, and ways to study the effect of introducing new technology into organizations.

The chapter covers the background to HCI, interaction with computers, computer-mediated communication, the psychology of computer use, models of human–computer interaction, computer system design and evaluation, and the social and organizational aspects of computer use.

A. Pioneers of HCI

HCI has been influenced greatly by a few visionaries who imagined new ways of working with computers. They should not be seen as isolated prophets. Their work was well known to later researchers and has had a strong influence on present-day methods of interaction with computers. Baecker and Buxton (1987) provide a valuable historical survey of HCI.

1. Vannevar Bush and the MEMEX

In 1945, Vannevar Bush, a scientific advisor to the United States government, had an article published in the *Atlantic Monthly* which argued that a successful peacetime research effort would depend on people having ready access to appropriate information. He proposed a future device called the MEMEX, which would extend the human memory by presenting and associating diverse pieces of information. An operator would call up text or images stored on microfilm and choose to follow trails of association to other related information or would add new trails to indicate a train of thought or a cluster of ideas. The MEMEX was never implemented, but the ideas—of using a machine to augment the intellect, of direct manipulation of information, and of trails of association—laid the foundations of multimedia computing.

2. Ivan Sutherland and Sketchpad

By the early 1960s, experimental time-sharing computers were being built that allowed a computer to work on several jobs simultaneously. Computer time could be switched automatically between users, allowing people to work at the computer screen in bursts of activity and solve problems by *interacting* with the computer rather than presenting it with pre-prepared programs. Early work on military computer systems had shown that it was possible to display and manipulate images on a computer console, and researchers began to explore the possibilities of graphical interaction between humans and computers.

Ivan Sutherland, a researcher at the MIT Lincoln Laboratory, implemented a drawing system named *Sketchpad* that demonstrated the power of pictorial interaction and introduced techniques, such as applying constraints to objects, still being developed for contemporary graphics packages.

3. Doug Engelbart and Augment

Doug Engelbart, as a graduate student at the University of California in the 1950s, advocated the computer as an "augmentation device," offering people new ways to study problems, experiment with ideas and hunches, and test possible solutions. Instead of being designed to solve a single problem, computer programs could be constructed as toolkits, with parts that could be reused and extended. The synergy that comes from combining these tools into an integrated "workshop" makes each tool considerably more valuable than if it were used alone, and the combined effort of people working together on the computer augments the abilities of individuals into computer-assisted communities.

With colleagues at Stanford Research Institute in the 1960s, Engelbart developed NLS (oNLine System, later called NLS/Augment), which assisted people in working together on tasks such as planning, analysis, and problem solving (Engelbart & English, 1988). It provided many novel facilities, such as active links between pieces of information, user-controlled windows, filters that displayed files with a specified content, the ability to send electronic mail, and shared-screen conferencing.

4. Alan Kay and the Dynabook

In the 1970s, Alan Kay wrote about an imagined self-contained knowledge manipulator in a portable package the size and shape of an ordinary notebook. It would be able to store and display text, drawings, music, and animations; and the owner would be able to design documents, compose music, create pictures, and communicate directly with other people through sound and image. Alan Kay named this notebook computer *The Dynabook* and in 1972 suggested that "The Dynabook is now within reach of current technology." As a member of the Xerox Palo Alto Research Center, Alan Kay was uniquely placed to realize his vision.

For 10 years he and colleagues developed a series of personal computers (which they called *interim Dynabooks*) and a computer language called *Smalltalk* to support the construction and manipulation of dynamic objects. In April 1981, Xerox announced the 8010 Star Information system, a personal computer designed for office use. Although the Star was a desktop machine rather than a portable package, it realized much of the Dynabook vision. The Star was the first commercial computer to be designed around a *graphical user interface,* which offered a consistent analogy to the top of an office desk with surrounding furniture and equipment. The objects on the simulated desktop were represented by pictorial *icons* indicating "in" and "out" baskets for electronic mail, file drawers, folders, and other electronic metaphors for objects in an office. Operations on the Star were carried out by *direct manipulation* of objects. For example, to file a document the user would move an icon representing the document over a picture of a file drawer.

The novel aspects of HCI found in the Star computer, such as the desktop metaphor, "windows," icons, and "pull down menus," were brought to the mass market in the Apple Macintosh computer. As well as providing a consistent "look and feel" to its range of computers, Apple set down guidelines for designers of Macintosh software which ensured that the Apple's investment in interface design and human–computer interaction would be reflected in software produced by other companies. By the mid-1980s, companies such as Apple, Microsoft, and Aldus were developing software for small business and home use that allowed people with no knowledge of computing and little training in use of the software to perform complex and highly interactive tasks such as producing spreadsheets and designing page layouts. The commitment of these companies to good practice in human–computer interaction, and their subsequent commercial success, has meant that research in HCI has an increasingly strong influence on commercial practice.

B. Interfaces

The interface to a computer is the combination of devices (such as a keyboard, mouse, light pen, screen display, and loudspeaker) that enable a user to interact with the computer. A main aim of interface design is to produce interfaces that hide the complexity of the computer, giving a user the impression of working directly on a productive task such as writing a document or creating an illustration. One way to do this is by presenting the interface as a *metaphor* for some more familiar system, such as a typewriter or set of illustrator's tools.

The advantage of an interface metaphor is that it provides a short-cut to learning a complex system, by offering concepts and operations that fit with the user's existing experience (such as "cutting" and "pasting" objects on the screen). Although metaphors play a central part in interface design, they can lead to faulty reasoning, when the interface does not match the familiar system. For example, in most interfaces cutting and pasting an object does not exactly match the physical process. When a screen object is cut, it "disappears" until the paste operation is performed. Metaphors can also be restrictive, tying the interface to concepts and operations that occur in the noncomputer world, rather than exploiting the possibilities of computer technology.

C. Interaction Devices

As computers are adapted for diverse tasks, from control of industrial processes to art and design, they are being fitted with *interaction devices* that enable objects on the screen to be created, selected, oriented, tracked, and moved. New input devices include trackballs, DataGloves, eyetrackers, digitizing tablets, and thumb wheels. The keyboard is still the main method of providing data to a computer, but the design of keyboards is changing to address concerns about fatigue and "repetitive strain injury." Some new keyboards include wrist supports and can split down the center, so that each half can be swung outward to keep the hand in a comfortable position.

Direct pointing devices such as a light pen, stylus, or finger on touch-screen can be moved directly to a point on the screen and can perform all the main interaction tasks, but writing on an upright screen can cause severe fatigue. A screen mounted at 30 to 45 degrees from the horizontal is more convenient both for pen input and for reading. Indirect pointing devices such as a mouse or trackball overcome the problem of writing on a vertical screen, but require hand–eye coordination and take up additional desk space. Coordination is not a major problem, however, and a child can learn to use a mouse in a matter of minutes.

There is much debate about the merits of different pointing devices, and factors affecting their use include the time required to select a near or distant target on the screen, ability to carry out fine movement for handwriting or drawing, muscular strain, and durability. Shneiderman (1992) proposes a touchscreen for durability in public-access applications, a mouse or trackball for accurate pointing, and a keyboard cursor when there are a small number of targets. Joysticks offer a firm grip and easy movement but are slow and inaccurate in guiding a cursor to a fixed destination.

A major limitation of all the devices just described is that they are intended for moving around a two-dimensional space on the computer screen. With the advent of three-dimensional simulations, shown on a computer screen or through helmet-mounted displays, there is a need for input devices with six degrees of movement (movement through three-dimensional space plus forward and sideways tilt and rotation). Such devices include the DataGlove (which fits over the hand and can register its position and gestures), the Polhemus tracker (a wand whose position and orientation in space is transmitted to the computer), and the spaceball (a small ball, mounted on a base, which can be twisted and pushed).

D. Communicating with a Computer

An influential account of human–computer communication is Norman's *execution-evaluation cycle* (Norman, 1986). The user of a system starts with a goal to perform some activity, forms an intention, and then specifies and performs some action. As the system responds, the user perceives the state of the system, interprets that state, and evaluates it with respect to the goals and intentions. This leads the user to set further goals and continue the interaction. The user's goals are expressed in psychological terms (such as "check the spelling of this word") but the system presents its current state in physical terms (such as a list of possible word corrections). The goals and the system state differ in form and content, creating "gulfs" that need to be bridged to ensure successful communication.

The *gulf of execution* represents the gap between a user's intentions and the means by which they can be carried out on the computer. The user specifies an appropriate sequence of actions and then carries them out (for example, by selecting an item from a menu) in a form that the computer can interpret. The *gulf of evaluation* is the gap between the computer's presentation of its current state and the user's expectation. To bridge that gap, a user must compare the state of the system, as presented on the computer screen or other output devices, with the original goals and intentions. If the computer does not appear to have satisfied the goal then the user must reformulate it (and possibly first attempt to undo the action that led to the wrong response).

A system designer can narrow the gulfs of execution and evaluation by such means as providing an interaction language that matches the user's intentions (for instance, by providing a spell checker, which can be applied to selected pieces of text and operated by a memorable and easily accessible physical act), by supplying interaction devices that allow a user to translate intentions into actions, and by presenting the system in a form that can be easily interpreted in terms of psychological goals. As an example of poor mapping in everyday objects, Norman describes a room with a panel of light switches where there is no simple relationship between the position of the switches and the position of the lights, so that it is necessary to discover by trial and error which switch controls which light.

For an interface controlling a power plant, a chemical installation, or an airplane, the difficulties and dangers can be considerably greater. Michie and Johnston (1984) argue that complex systems need a "human window" that, in an emergency, can give the human operator an intelligible précis of the situation. However, the term *window* is highly misleading; a monitoring computer is not a sheet of glass but one complex system interpreting another complex system to the human user. Adding a program that summarizes the data may give an appearance of clarity but cause human controllers to lose contact with the real world and, at worst, treat a crisis as a computer game.

E. Styles of Communication

The cycle of activity and response creates a dialogue between the human user and the computer that has some similarity with human-to-human conversation. Successful communication depends on establishing a shared representation and holding a dialogue to carry out a task.

1. Command Line Interaction

The earliest interactive computers communicated via commands typed at a console and responded with printed output. Command line interaction is still found on powerful environments such as the UNIX operating system, because it gives the user direct access to computer operations. These can be strung together into "scripts" to carry out multiple tasks, with the output from one operation being used as input to the next. The penalty for this flexibility and power is that the computer offers little assistance as a conversational partner. The user must remember the commands and the syntax required to combine them into scripts. Command line interaction may be suitable for experienced "power users" of a general-purpose computing environment, but in many tasks, such as drawing, issuing commands to the computer is tedious and unintuitive.

2. Menus and WIMP Interfaces

Menus pack together commands into a list from which one or more can be selected. Styles of menu include pull down (where selecting a header word causes a menu to appear below it), pop up (where the menu appears beside an item selected on the screen), walking (where selecting a menu item can cause a submenu to appear beside it), and pie (where the menu radiates out from a central point). Menus overcome some of the learning and memory problems of command line interfaces but do not offer the power of combining commands into programlike scripts.

Menus form a part of WIMP (*windows, icons, menus, pointers*) interfaces. A WIMP screen displays a number of overlapping, bounded windows and a pointing device such as a mouse moves a cursor to one of the windows. Menus allow commands to be directed to the selected window and icons can represent commands or objects (such as "closed" windows) in pictorial form. Other screen elements, such as "buttons," "palettes," and "dialogue boxes" allow the user to communicate in a variety of modes.

3. Natural Language

Some tasks, such as dictating a memo, would clearly be made easier by speaking to the computer and having the spoken word translated directly into text. Speech input may also be valuable for public-access systems, such as timetable enquiries or tourist information, where there is limited, information-seeking communication. Early natural language interfaces were limited to single word input and had to be trained to recognize the speaker's voice and intonation. More recent systems can recognize continuous, slowly spoken speech and require less or no training. Natural language offers new forms of interacting with machines, such as by telephone conversation, but the apparent ease of spoken conversation hides the difficulty of conducting a useful dialogue. A computer program is designed to perform a restricted task, such as giving tourist information, and cannot behave like a human conversational partner. Either the user must adapt to the limited linguistic capabilities of the computer (by guessing what language forms the computer might recognize and rephrasing commands until they are accepted) or the computer must direct the dialogue (leaving the user to give limited responses such as yes, no, or numerals).

4. Pen Input

Software for the automatic recognition of handwriting has encouraged the development of pen interfaces, which allow informal communication with the computer through writing, sketching, and gesturing. Software to recognize cursive (joined-up) handwriting is still slow and unreliable but good enough to provide an interface to *personal digital assistants,* or PDAs, which combine the facilities of a diary or personal organizer with communication by fax or electronic mail. A form displayed on the screen can be filled out by writing responses with a stylus that are then converted into digits or words. A handdrawn diagram can be tidied by converting roughly drawn shapes into exact straight lines, circles, and boxes. A gesture, such as moving the stylus back and forth over an object, can be interpreted as a command, such as "delete this object." Pen input is slower than typing for simple text input, but it opens possibilities for "informal interaction" with the computer through sketches and gestures.

5. Direct Manipulation

The term *direct manipulation* describes the alteration—by pressing, dragging, and reshaping—of objects displayed on a screen or in a simulated "visual world." Instead of commanding the computer to perform an action, the user performs the action directly on the simulated object. *Virtual reality,* in which the user is placed in a simulated world presented on a screen or through helmets with miniature displays for each eye, is an extreme form of direct manipulation, where the objects in the simulation can be manipulated as if they were in the real world. But direct manipulation is not restricted to everyday objects. The same interaction techniques can be used to operate on diagrams, charts, and documents. Direct manipulation can provide the user with a rapid response to actions and can lessen the gulf of execution and evaluation by allowing the system state to be changed directly, rather than obliquely through commands.

Direct manipulation allows many tasks to be performed quickly and naturally, but it lacks the expressiveness and combinatorial power of language. Even the restricted syntax of command line interfaces allows new expressions to be constructed by combining primitive commands, and natural language offers the subtlety of referring to classes of objects, indicating conditionality ("if the diagram does not fit the screen then show me the center portion"), and indirect reference ("are there any objects that have the following properties . . . ?").

New methods of communication can combine the immediacy of direct manipulation and the expressiveness of natural language. These include *programming by example,* where the user performs a sequence of operations and the machine records them as a generalized procedure, and direct communication with *agents,* which perform complex, general tasks (such as arranging a meeting, or searching multiple databases) on behalf of the user.

F. Communicating Through the Computer

As computers became linked through *local area networks,* providing communication between machines in a building or locality, and *wide area networks,* such as the Internet, which connects computers throughout the world, it became possible to use the computer as a communication device between people. *Asynchronous* connection, such as by electronic mail, is still the main method of communicating between computer users, but techniques for compressing data and high-bandwidth data lines now allow direct, *synchronous,* communication from computer to computer by voice and moving image. The styles of communication range from *videophones* to *awareness* systems (where windows on the screen give an impression of activities in other offices or buildings).

The computer acts as a *mediator* of person-to-person communication. It may be designed to be unobtrusive, as in a *video tunnel* with a camera directly behind the computer screen, so that users can converse and hold eye contact as if face to face, or it may offer a *shared medium,* such as a space on the screen where two or more users can draw and write. The study of *computer-supported cooperative work* is concerned with the design and use of computers to facilitate shared working.

II. THE PSYCHOLOGY OF COMPUTER USE

Most interactive computer systems are designed to engage the mind, so cognitive psychology has the potential to assist in their design and evaluation. Surprisingly few psychological theories, however, make predictions about human–computer interaction or can be used directly to guide design. The findings of cognitive psychology have generally been used either as broad (and often inaccurate) guidelines, such as "show seven plus or minus two items on the screen," or as general frameworks. For example, work on mental models leads to the design of systems based on deliberate metaphor (such as the desktop metaphor of the Apple Macintosh interface).

Landauer, in an influential paper, "Relations between Cognitive Psychology and Computer System Design" (1987), suggests four principal ways in which cognitive psychology can interact with computer system invention and design:

1. We may apply existing knowledge and theory directly to design problems.
2. We may apply psychological ideas and theories to the creation of new models, analyses, and engineering tools.
3. We may apply methods of empirical research and data analysis to the evaluation of désigns, design alternatives, and design principles.
4. We can use problems encountered in design to inform and guide our research into the fundamentals of mental life.

A. Applying Psychology to Design

A computer system that interacts with a human user should take account of the properties and limitations of the human mind and body. The nature of the human mind has been studied by successive generations of psychologists, and some results of this work are directly applicable to system design.

1. Memory

Findings from research on human memory that could influence computer system design include recency and primacy effects, chunking, and the associativity of semantic memory.

Information in short-term memory decays rapidly. If items are presented in serial order, those items toward the end of the series will be remembered well for a short period of time (the *recency* effect); items at the start of the series will be remembered well for a longer period of time (the *primacy* effect); and items in the middle of the list will be less well recalled after a short or longer delay. This suggests that list presentations, such as pull-down screen menus, should be organized so that less important items are placed in the middle of the list or, if all items are important, then the user should be given assistance in recall and selection.

Miller's (1956) work suggests that human retention of short-term information is limited to around seven *meaningful chunks*. Thus, if information on a computer screen can be grouped into meaningful chunks, then it is likely to be more memorable than as disparate items. As a simple example, presenting the time as 19:55:32 is likely to be more memorable than as 195532. Considerable work has been done on how to present computer commands in ways that are meaningful and well-structured. Suggested guidelines (from Shneiderman, 1992) include the following: choose names for commands that are meaningful and distinctive, group the commands into easily understood categories, present the commands in a consistent format such as action–object (e.g., "delete word"), support consistent rules of abbreviation (preferably truncation to one letter), offer frequent users the ability to create "macros" that group a series of commands, limit the number of commands and the ways of accomplishing a task.

The final guideline seems counterintuitive, and certainly limiting the commands and functions too far could impede use, but *interference effects* can occur when there are alternative ways to perform tasks. For example, in Microsoft Word a selected piece of text can be turned to italic by (among other ways) selecting "italic" from the "format" menu, pressing the COMMAND–*i* keys, or pressing COMMAND–shift–*i*. Some of these methods work with other word processors; some do not or they invoke other commands. Ensuring consistency of appearance and operation within a program and across related programs is a major aim of interface design. Manufacturers such as Apple, IBM, and Sun Microsystems provide detailed *interface guidelines* to developers of software for their products, to ensure a consistent *look and feel* and to set standards for command names and methods of interaction.

Long-term memory appears to be organized around concepts sharing similar properties and related by association. This *associativity* can be mirrored in the computer through *hypermedia* systems that present information as texts or images, with explicit links to associated items. Thus, a hypermedia guide to an art gallery might display a painting on the screen and provide buttons allowing the user to display other paintings by the same artist, or in the same style, or on display in the same room. The computer can also offer aids to learning or invention by providing *external representations* of associative memory, allowing learner, writer, or designer to set down ideas as visual notes on the screen and to associate them by drawing labeled links.

2. Perception and Attention

A computer assaults the senses and demands a high level of attention. Unlike a book, it has an active light-emitting display; unlike a television, it requires the user to sit close to the screen and interact with it. Helmet-mounted virtual-reality displays have been claimed to cause fatigue and disorientation after 10 minutes of use. The ability of the computer to abuse the senses means that special care should be given to designing interfaces that complement human perception. *Visual acuity* is the ability to distinguish fine detail. Acuity increases with brightness so, in general, a bright screen will be more readable than a dark one, and dark characters on a white background will be more readable than light characters on a dark background. But high luminance displays are perceived to flicker, and since flicker is more noticeable in peripheral vision, a larger screen will appear to flicker more.

The choice of color for displays is fraught with difficulty. When color displays first appeared, programmers splattered primary colours on the screen, hoping to make them more attractive, but generally causing confusion and headache. Different colors are perceived as having different depths (for example, red appears to stand out, whereas blue recedes) and strong differences in visual depth can be fatiguing and distracting. Blue lines on a black background, or yellow lines on a white background, are hard to distinguish. Eight percent of the European male population is colorblind and cannot distinguish red from green. In general, color should be used sparingly, and for a purpose, such as to distinguish items in a list or to indicate important information.

Maguire (1990) collated the following guidelines for the use of color in displays:

1. A neutral color helps set off a full color.
2. Colors near in hue (on the hue circle) form pleasing combinations and will lend a predominant tone.
3. Complementary colors (opposite on the hue circle) contrast and give a warm–cool effect.
4. Colors that are 120° apart on the hue circle are easy to distinguish but often appear garish.
5. Color edging enhances polygons.
6. Common denominations of colors should be used where possible. For example, red: danger or stop, yellow: caution or slow, green: OK or go.

Sound has been little used in interfaces, other than to provide warnings. However, experiments on *auditory icons* (which simulate everyday sounds in computer simulations of, for example, industrial processes) and on *earcons* (which use structured combinations of notes to represent actions and objects), suggest that sounds can be effective in complementing visual presentations and in monitoring of off-screen processes. Dix, Finlay, Abowd, and Beale (1993) provide a good overview of multisensory systems.

3. Motor Skills and Selection

Studies of reaction time have some application to computer design, particularly in computer interfaces to machinery, but most interaction with computers is premeditated (the user forms an intention and then specifies and performs an action), so considerations of speed and accuracy are generally more important. Fitts' law (1954) is a good predictor of the time to move a given distance to a target of a specified size; it takes longer to point to more distant and smaller targets. This suggests that, in general, targets on the screen such as buttons should be large and close to the user's start point.

Landauer (1987) combines Fitts' law with Hick's law to predict the time a user takes to find an item in a hierarchical menu (a menu with two or more levels). Hick's law states that the mean response time in simple decision tasks is a linear function of the transmitted information. This suggests that menus with more items per level will require a longer time to select an item; and Fitts' law suggests that more alternatives will require targets that are smaller and harder to select. But the total search time is equal to the mean time per step, multiplied by the number of steps down the menu hierarchy. A menu with many items at each level but fewer levels will have a longer choice time, but fewer steps. By combining the equations, Landauer showed that search time will be minimized by using menus with as few levels and as many items as are feasible within the constraints of space and display. Studies of menu-selection tasks broadly support the finding, and this is one of the few direct applications of experimental psychology to interface design.

4. Skill Acquisition

An experienced user of a computer system will not perform the same actions and require the same assistance as a novice. Computer use is an acquired skill, and studies of skill acquisition suggest that a learner progresses from learning facts and applying general procedures, to acquiring directly applicable and more specific procedures, and finally to automatic execution of tasks. A expert in a skill such as chess does not consider many alternatives and make explicit plans, to the extent that a novice might, but instead relies on a meaningful pattern of pieces to prompt a good move.

The implications for human–computer interaction are that novice computer users should be offered help in learning the basic objects and actions of the system, through tutorials and context-specific help. The Macintosh interface, for example, offers *balloon help,* whereby "balloons" offering simple advice pop up over objects on the screen. As they gain expertise, users need practice and assistance in forming a *command set* of the most useful commands (which will depend on the user's needs and abilities). Experienced *power users* can be offered ways to reduce time and effort by, for example, invoking commands directly from the keyboard rather than using a mouse and menu.

B. Using Psychology to Create New Models

The earliest use of computers was to solve specific problems, in mathematics, engineering, ballistics, logic. Accordingly, the research psychology of problem-solving was applied and adapted to the study of computer use. Newell and Simon's (1972) "information processing psychology" both described human problem solving in information-processing terms and was applied to the study of problem solving with computers. It led directly to the development of cognitive models of problem solving, from high-level descriptions of the writing process (Flower & Hayes, 1980) to a keystroke-level model of interaction (Card, Moran, & Newell, 1983). These models have been used to predict the behavior of users solving problems with computers and to inform the design of more effective and comprehensive computer systems.

More recently, it has been recognized that people use computers for tasks that do not easily fit the problem-solving mold, such as design, exploration, and communication. New models are being developed that draw on studies of human creativity, exploration, and social interaction. The modeling of human–computer interaction is discussed in Section III.

C. Applying Psychology to Evaluation

The computer is a piece of equipment, and many of the experimental psychology techniques for evaluating human interaction with equipment are applicable to human–computer interaction. For example, Reece (1993) has carried out an elegant investigation, using standard experimental methods, to compare the relative merits of pen and paper, a computer word processor, and a simulated speech-driven word processor. Shneiderman (1992, p. 18) indicates five measurable human factors that are central to evaluation: time to learn, speed of performance, rate of errors by users, retention over time, subjective satisfaction.

Where the computer differs from everyday equipment is in its flexibility of function and appearance, resulting in a large space of possible system designs and presentations. Much of the work in human–computer evaluation is not concerned with testing clear hypotheses and comparing fixed products, but with assessing the quality of early designs. New techniques have been devised to evaluate prototypes and design possibilities rather than finished systems. These include *Wizard of Oz* evaluation (where a human pretends to be the computer, for example, by mimicking a speech-understanding system) and *design space analysis* (for organizing and comparing alternative designs). Section V covers the topic of system evaluation in more detail.

D. Applying Studies of Human–Computer Interaction to Psychology

The versatility of the computer makes it a valuable device for studying the psychology of learning and problem solving. As well as providing a tool for problem solving and a medium for communication, it can automatically record data and analyze patterns of interactions.

The study of children as programmers has provided insight into children's development of problem-solving skills, such as the use of physical devices like the Logo "turtle" to catalyze understanding (Papert, 1980).

Investigations of programming as a skill have revealed differences between the problem-solving behavior of novices and experts (Jeffries, Turner, Polson, & Atwood, 1981). Both novices and experts tend to start from a general problem goal and refine it into subgoals and down to lines of program code. But novices tend to expand one part of the problem down to its lowest level before starting on the next part, whereas experts generally consider the entire problem before developing it to a deeper level.

Computer games have formed the basis of studies of motivation (Malone, 1981) and collaboration and conflict (Hewitt, Gilbert, Jirotka, & Wilbur, 1990).

As people have begun to use computers as an everyday tool to augment their intellect, to extend their memory, and to distribute cognition among a group of coworkers, this has led to the study of computer-augmented and computer-mediated cognition.

III. MODELING HUMAN–COMPUTER INTERACTION

Models of human–computer interaction serve much the same purposes as architectural, scientific, or engineering models. They can be used to predict behavior, assist in design, and evaluate competing theories and designs. Where they differ is in the importance given to modeling human cognition. Understanding the psychology of the computer user is important in creating systems that solve interesting problems, respond in appropriate ways, and are engaging and easy to use.

There has been much confusion in the literature about the types, names, and purposes of models of HCI. Young (1983) attempted to sort out the mess by suggesting two types of models—models of the user and models of the computer—that may be held by different entities: designers, researchers, computer systems, and users. Thus, a researcher may develop a model of a user to understand the psychology of computer use or a computer may hold a rudimentary model of its user to offer individualized help or guidance. The models with most significance to HCI are the designer's model of the computer, the designer's model of the user, and the user's model of the computer.

A. The Designer's Model of the Computer

A software designer has direct control over the form and function of the software but not of the people who use it, so designers' models of the computer tend to be more detailed and formal than their models of its users. Software descriptions can be divided broadly into *syntactic models,* concerned with the structure of the dialogue between user and machine, and *semantic models,* which describe what the user's actions do. The simplest syntactic models describe the desired states of the computer system and transitions

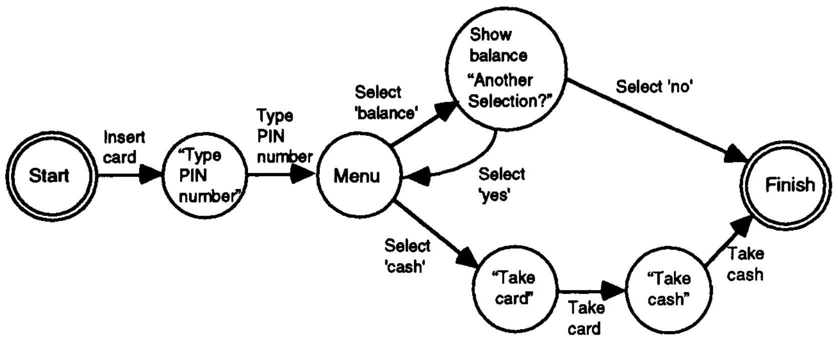

FIGURE 1 A state transition diagram for a simplified automated teller machine.

between them. They can be represented as *state transition diagrams,* with each state shown as a circle and each transition as a labeled, arrowed line. The transitions correspond to actions performed by the user, such as selecting from a menu or pressing a key. Figure 1 shows a state transition diagram for a simplified bank automated teller machine.

A state transition network is easy to create and understand, but for a large system it can become complex and tangled. A *hierarchical state transition network* deals with complexity by showing subsystems as "boxes" in the diagram, where each box can be described as a separate diagram. Another problem is that the diagrams do not show events and dialogues that can occur concurrently or that interrupt each other, such as might appear in a computer-aided design package with concurrently active windows, menus, palettes, and forms. Harel's (1988) *state charts* provide a way of showing complex, concurrent systems as diagrams.

An alternative way of representing the syntax of computer systems is as formal grammars or rules. The *BNF* notation is typically used to describe the structure of programming languages, but has been applied to HCI. Condition–action rules, or *production rules,* of the form

IF (condition) THEN (action)

can specify events which change the system state or cause program actions. For example, the rule

Insert card → <display "Type PIN number"> Wait-for-PIN

indicates that the event of inserting a card causes the ATM to display the message "Type PIN number" and puts it into a state of waiting for a PIN number to be typed. Production rules are better than state transition networks at describing concurrent events but less appropriate for specifying sequences of events.

Semantic models of interaction with the computer specify not just the transitions between states of the computer system but also what functions are invoked at each transition. The most straightforward way is to add functional descriptions to syntactic representations such as state transition diagrams or production rules. These could be specified in a notation of formal semantics or simply as calls to programming language procedures. More sophisticated formalisms such as Z (Spivey, 1988) provide a complete formal notation to describe the semantics of programs. They were developed to reason about the behavior of programs but have been adapted for producing precise specifications of interactive software. Producing a formal description of an interactive system can be difficult and time consuming, and it may not tell what a designer most wants to know, such as "will the program be easy and intuitive to use?"

B. The Designer's Model of the User

The ideal user, as far as the software designer is concerned, is someone who behaves in a predictable manner, forming clear goals and then carrying them out in a series of steps that can be programmed as basic operations for the computer. In practice, no user behaves in this way. Real users are unpredictable, have conflicting goals (or may not even be able to express their needs in terms of clear goals), change their minds, make mistakes, and may end up muddled and maddened. Not surprisingly, it is easier to model an ideal user than a real one. An idealized model can help the designer to produce software that performs well with competent users, but too little attention has been given in HCI to individual differences in software use.

The most influential approach to modeling the user is Card et al.'s (1983) GOMS (goals, operators, methods, and selection rules) method. GOMS characterizes a user of a computer system as an information processor, with interacting perceptual, motor, and cognitive subsystems. It assumes that the user acts rationally by forming *goals* that define some state to be achieved. A goal can be achieved by one or more alternative *methods,* where a method is a sequence of elementary perceptual, motor, or cognitive *operations* (such as "Move cursor over word," "Double click mouse button," "Press delete key"). *Selection rules* choose between alternative methods to accomplish a goal. Using the GOMS formalism, a high-level goal, such as "edit manuscript" can be broken down into a sequence of methods and operators, where each operation can be provided by an elemental computer procedure.

GOMS can make some broad predictions of human behavior. The depth of the subgoals gives an indication of the load on a user's short term memory, and the length of the sequence of operations provides a rough measure of the time taken to carry out a method.

Card, Moran, and Newell (1980) demonstrated the power of GOMS by using it to predict the time expert users will take to execute tasks on a computer. They proposed that, at the level of individual motor actions, the time taken to execute a computer task can be described as the sum of the times taken in keystroking, pointing (e.g., with a mouse), homing (on a target), drawing, thinking (by the user), and responding (by the system). They encoded methods, such as editing text with keyboard and mouse-operated editors, in the GOMS notation and produced estimates of individual operations by means such as Fitts' law. This enabled them to predict the times taken to carry out tasks such as replacing a five-letter word with another five-letter word. Comparing their predictions with the performance of human users, they found that the time required for experts to perform a unit task can be predicted to within 20%.

Kieras and Polson (1985) clarified and enriched the GOMS model by providing two parallel descriptions, of the user and the computer. The user's knowledge is expressed as condition–action rules where the condition can match goals, information in the user's memory, or information provided by the computer. If a pattern matches successfully, the action part of the rule is performed, to alter the representation of the user's memory or carry out some operation on the computer. For example, a simple rule for moving the cursor to a given line might be (from Dix et al., 1993, p. 199):

```
(MOVE-UP
IF      (AND  (TEST-GOAL move-up
              (TEST-CURSOR-BELOW  %LINE)  )
THEN (        (DO-KEYSTROKE 'K')  )  )
```

This means that if the user has the goal of "move-up" and the current position of the cursor is below the desired line on the screen, then the user should perform the operation of pressing key *k* (which in the particular text editor moves the cursor up a line). This rule would "fire" repeatedly so long as the condition is satisfied; that is, until the cursor is at the desired line. To describe the states and operations of the computer, Kieras and Poulson use a form of state transition network. Combining a model of the user with one of the computer system enables the method to describe mappings and mismatches between a user's goals and the provisions of the computer.

GOMS and its associates offer fine-grain accounts of a user's interaction with the computer. But, particularly in the early stages of design, a system designer may be attempting to understand the task and choose between alternative designs. In that case the designer needs a more general model of problem solving and user behavior which can help to ensure that the software is comprehensive and suited to a broad range of users.

Distributed cognition (Hutchins, 1990) provides an account of interactions beyond the individual. A *functional system* is the collection of human participants, computer systems, and other objects used in the work practice. Functional systems that have been studied include air traffic control, computer programming teams, and civil engineering practices. The aim of the distributed cognition approach is to analyze how the different components of the functional system are coordinated. This involves studying how information is propagated through the system and how work is maintained by the participants adjusting their activities in synchronization with each other.

C. The User's Model of the Computer

People spontaneously employ *mental models* to assist their understanding of complex systems. Mental models are simplifications, often in terms of a metaphor, such as a word processor as a "computer typewriter." They frame our understanding of the world and enable people to operate and make predictions about new technology, by relating it to prior experience.

The term *system image* (Norman, 1986) describes the user's conception of a computer system, which may be in terms of a metaphor or a set of functions. If the user is given a system image before interacting with the system, then the image will determine how the user conceives and begins to operate the equipment. Designers can draw on mental models by designing software to fit a metaphor. Thus, electronic mail has deliberate similarities to the postal system, with "mail boxes," "addresses," and "postmasters." A system image need not represent the way the system is actually constructed, but a well-chosen system image should enable the user to interact with the computer in a way that meets with expectations. It allows the user to make reliable predictions about the system's behavior and be satisfied that the system is not going to carry out unexpected actions.

So long as the computer works according to the system image then the user is led to treat it as a familiar object. But, if the system image ever fails, then the user is left stranded, without the depth of knowledge needed to repair the problem. For example, if a computer network fails to deliver an electronic mail message, then the message is usually returned to the sender with a detailed diagnosis of the error. The diagnostic information is given in terms of gateway computers and message handling programs, far from the image of electronic mail as a postal system. Studies of breakdowns in human–computer interaction, and of the user's subsequent attempts at repair, can be valuable in revealing people's (sometimes surprising) mental models of technology (Sharples, 1993).

IV. SYSTEM DEVELOPMENT

The conventional approach to developing a computer system is to split the process into a number of discrete stages, leading up to the finished working product. Definitions of the stages vary, but they usually include the following:

Requirements analysis, which describes the purpose of the system along with the main tasks it has to perform.

System design or *functional specification,* which specifies how the tasks are to be performed, and breaks the system into components that need to be either programmed or adapted from existing software.

Detailed design, where each component of the system is described in sufficient detail that a programmer can code it.

Implementation, which must be done in a suitable programming language.

Integration and testing, which is done to the different pieces of program.

Maintenance of the system, by correcting errors and updating the software to cope with changes in hardware and requirements.

This staged or *waterfall* (or cascade) approach was introduced to assist the development of large corporate systems such as payroll packages, which typically have tight specifications, large teams of programmers, and very little interactivity. A major industry has grown around the waterfall method of software development, with structured design methods such as JSD (Jackson structured design) and SSADM (structured systems analysis and design methodology) promoting a strict discipline of design and testing. The basic principles, of ensuring that the software meets requirements and that it should be well integrated and tested, apply equally to payroll packages and painting programs. But the main difference is that the operation of highly interactive systems cannot be fully specified in advance of implementation. Any reasonably complex interactive program will be used in ways unforeseen by its designers. For example, spreadsheet programs were originally introduced to ease the job of repetitious calculation. But since their introduction, users have adopted spreadsheets for forecasting, visual presentation of information, database management, producing timetables, and many other purposes.

User-centered design is a general term for a design process that considers the needs of users in the design of interactive systems. User-centered design relies on understanding the users and their interaction with technology. Designers must respond sympathetically to the, often conflicting, demands of users while maintaining the integrity of the system. Not surprisingly, attempts have been made to reconcile the two approaches to design, by adapting the waterfall model to the design of interactive systems.

A. Adaptations of the Conventional Software Development Method

Usability engineering provides information to guide software development by specifying *usability metrics* that the finished system should satisfy. *Usability metrics* are normally quantitative measures in the areas of *learnability* (the time and effort needed for learning to use the system), *throughput* (the number of tasks accomplished in a given time and the errors made), *flexibility* (the extent to which users can adapt the system to new ways of working as they become more experienced in its use), and *attitude* (the attitude of the users toward the system). The measures, such as "time taken to install the system," can be specified in advance as part of the system design to provide guidelines for the programmers.

Usability metrics are only one type of guideline. Other, more qualitative guidelines specify general attributes of the system, such as the colors to be used or the appearance of objects on the screen. They can be very specific, for example indicating the phrases to be used in error messages. At their best, guidelines reflect good design practice, match what is known about the needs of the users, and give the software a consistent "look and feel." The major software manufacturers issue books of guidelines to their software developers, such as Apple's *Human Interface Guidelines* (Apple Computer Inc., 1987).

User interface design environments automate or assist the work of designing user interfaces by providing a kit of "interface parts" such as buttons, menus, and forms, each with appropriate functionality, which can be fitted together into a complete user interface. Each of the components adheres to interface guidelines, so a design environment can provide a short-cut to design and an assurance of basic quality.

B. Departures from the Conventional Software Development Method

The departures from the conventional method of software development can be classed under the general heading of *user-centered design,* since they give high priority to the needs of the eventual users. They do this either by analyzing the tasks that typical users will perform or by involving actual users in the design process. The second approach is the more radical because it requires an empirical "trial and error" process, with designers testing out early prototypes of the software on users and making repeated revisions to the plans. In terms of the waterfall model of software design, the later stages of the design process provide information to the designer that may result in revisions of the earlier stages and a series of evolving prototype systems. In some cases, such as the development of systems for casual use in a public area, such as a tourist information center, users may be recruited as part of the design team, offering advice and criticizing mockups and prototypes.

Task analysis is the study of what people do when they carry out tasks. It differs from user models such as GOMS in that the emphasis is on observable behavior, rather what might be happening in the user's mind. Task analysis has a long history in the study of work behavior. Its more modern variants, such as *TAKD* (task analysis for knowledge description; Diaper,

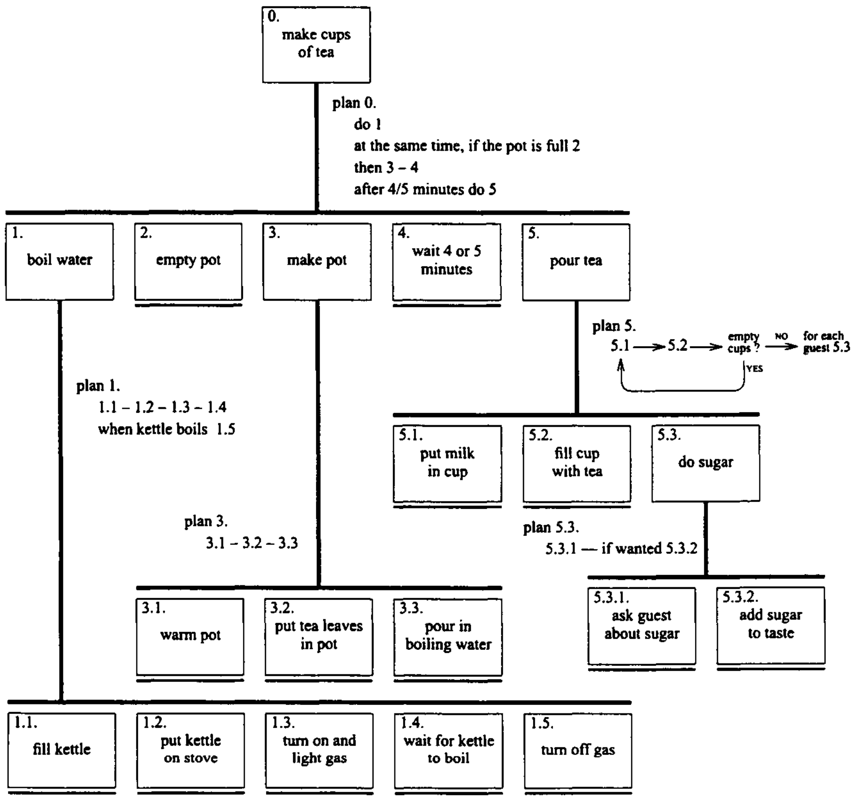

FIGURE 2 Task hierarchy for making cups of tea (from Dix et al., 1993)

1989), combine a behavioral description with an analysis of the knowledge people require to perform the task. The start of a task analysis is usually to break down an observed task into a hierarchy of subtasks, accompanied by plans that describe in what order and under what conditions the subtasks are performed. Figure 2 shows a task hierarchy for making cups of tea (from Dix et al., 1993). The analysis may then go on to produce taxonomies of the objects and actions involved in the task, accounts of the goals of a person engaged in the task and plans for accomplishing a goal.

To carry out a task analysis requires detailed observation and recording of people engaged in appropriate activities. A video recording can reveal the structure and sequence of activities, but other methods are needed to infer a user's goals and plans. One general and useful technique is *protocol analysis*. A person is asked to talk about his or her intentions and plans at the same time as performing the task. These *concurrent verbal protocols* are recorded and matched against the activities to give an understanding of not only *what* activity is being performed, but also *why* it is being carried out, for what purpose and to achieve what goal.

Task analysis can show the structure and sequence of a task, such as composing a document or designing a building, that could be supported by a computer. It can also be carried out for people working with computers, to measure usability or to study problems and breakdowns in performance. The weakness of task analysis is that it can be time consuming to prepare, and although it may reveal the structure of the task, it may not be much help in deciding how to design a system that supports and augments that task.

Rapid prototyping is one of a number of methods that involve eventual users in the early stages of design. One or more mockups or prototypes are constructed that exemplify some aspects of the system, such as alternative interfaces or different presentations of the data. The prototypes may be quite simple. They could be in the form of sketches of an interface, "storyboards" or "slideshows" to show a sequence of actions. These prototypes are shown to users, for comment, as part of systematic surveys and comparisons, or to raise users' understanding and enable them to contribute to further design. Rapid prototyping environments, such as HyperCard, allow screen displays to be constructed rapidly using screen drawing tools. They can then be given limited functions, such as allowing objects on the screen to become buttons that, when pressed, cause a move to another screen display. The original rapid prototyping tools required the mockups to be thrown away once they had been tested. But more recent development environments offer more powerful programming languages, allowing the mockups and early prototypes to be developed into full systems.

Rapid prototyping is one method employed in the *iterative* approach to software development. This maintains that, unlike conventional software development, the design of interactive systems is not primarily concerned with specifying the functions and input–output behavior of the computer, but with fitting the computer into the plans and interleaved activities of a human user. Not only are the plans and actions of users of paramount importance to the software designer, they are inherently ill-defined. The art of iterative system design is to *anticipate* the many ways in which the system might be used and *adapt* the system to suit the users' needs as they become identified.

V. EVALUATION

Evaluation of HCI is concerned with making judgments and measurements of the usability and potential usability of interactive computer systems. An evaluation can be carried out at any point in the software life cycle: to choose between competing requirements, to decide what features and functions should be included in a proposed system, to validate the system with respect to the requirements, to verify that the system meets the specifications to compare different implementations, or to provide metrics such as learnability, ease of use, or speed of operation. The type of evaluation can range from formal measures of usability (for example, to ensure that the system meets industry standards) to offering the system for informal comment.

The Open University Guide to Usability (The Open University, 1990) classifies evaluation methods into five categories:

• *Analytic evaluation* uses semi-formal methods, such as task analysis, to predict the performance of expert users of a system. It allows designers to predict the performance of users in terms of the physical activities and cognitive operations they must carry out to perform a given task. It can be applied early in the development process, reducing the need for iterative prototyping. But it should be used with care, since it is intended to model the performance of an ideal user carrying out tasks without error.

• *Expert evaluation* calls on people who are experienced in interface design or human factors to make judgments about the usability of the system and to suggest improvements. This can range from demonstrating the system to colleagues for informal comment to presenting a set of design issues that need to be resolved.

• *Observational evaluation* involves collecting data on how users interact with a prototype of a finished system. It may be a controlled study, carried out in a usability laboratory, or a naturalistic study, with the system set up in the workplace of a potential user. The data can come from direct observation and field notes, video recordings, automatic logging of the user's interactions, or analysis of verbal protocols.

• *Survey evaluation* asks users to give their opinions of a system that they have used, through a structured interview or questionnaire.

• *Experimental evaluation* requires the investigator to form hypotheses about the interaction, which are then tested using the methods of experimental psychology. Thus, one might compare the time taken to perform a given task, such as editing a document, using two different interfaces.

The different approaches to evaluation reflect the differing paradigms of psychology and come laden with prejudice and anecdotal evidence about the "best" way to develop and assess interactive software.

A team from Hewlett-Packard laboratories (Jeffries, Miller, Wharton, & Uyeda, 1991) performed a careful comparison of four methods of user interface evaluation: heuristic evaluation, software guidelines, cognitive walkthroughs, and usability testing. Heuristic evaluation is similar to "expert evaluation," where specialists in user interfaces study the interface in depth and look for properties that they know, from experience, will lead to usability problems. The use of published guidelines is not strictly a method of evaluation, but it does provide specific recommendations about interface design that can help to identify problems. A cognitive walkthrough is a type of analytic evaluation, where the developers of an interface carry out a series of tasks that a typical user might perform. They compare the actions of the interface with the users' goals and knowledge and record discrepancies between the users' expectations and the steps required by the interface. Usability testing begins by analyzing the users' needs and setting usability criteria for the system. Experimental evaluation is then used to test whether the system meets the criteria.

The comparison of the different techniques found that heuristic evaluation produced the best results. Heuristic evaluation uncovered the most problems with the interface (50% of the known problems compared with around 14% for each of the other methods) and at the least cost in terms of time. The method does, however, depend on finding several people with the knowledge and experience necessary to apply the technique.

The main divergence in approach to evaluation is between laboratory studies, carried out in a *usability laboratory* under carefully controlled conditions, and *situated studies* of people interacting with computer systems in their normal working environment.

The main advantage of usability engineering is that it sets explicit measures of usability (such as the time taken to perform a task, the number of errors, the number of commands used, and the ratio of successful completions to failures), which can be used to judge the value of the system once it is finished. The concomitant problem is that the measures may not capture the real qualities and failings of the system and may divert the designer into fixing minor defects rather than considering alternative solutions. For example, usability measures applied to the design of a word processor may help in rearranging items on a menu (for some word processors this is equivalent to rearranging deck-chairs on the Titanic), but they do not address the deeper issues of "how do people write?" and "what computer tools can best support the writing process?"

Situated studies can call on the techniques of ethnography and social psychology to provide a rich picture of the ways in which technology mediates the activity of individuals and groups. They can reveal breakdowns in understanding or communication caused by system failure or human misconception, and they can suggest new areas of concern to a system developer (such as the importance of tactile feedback). But the amount of data collected from such studies can be overwhelming, and there may be no guarantee that a detailed investigation of one working environment can be applied to the technology set in a different situation.

The best conclusion is that usability measures and situated studies can inform system design, but there is no substitute for the heuristic knowledge and experience of experts in HCI.

VI. BEYOND HUMAN–COMPUTER INTERACTION

The phrase *human–computer interaction* invokes an image of a person sitting alone at a console, communing with a computer program. That image fits only one aspect of computer use and is becoming increasingly inappropriate. Lawrence Tesler (1991), in one of a perceptive series of articles on computer networks in *Scientific American,* describes four paradigms of computing—batch, time sharing, desktop, and network—associated with the four decades from the 1960s to 1990s (see Table 1).

The 1990s is seeing the emergence of fully mobile computers, in communication by wireless networks. They can integrate communication services (such as fax and electronic mail), personal organizers (such as diaries and meeting schedulers), and work tools (such as document processors and expert systems). Instead of interacting with a computer, a person will work with many tools and applications, situated on a number of computers

TABLE 1 The Four Paradigms of Computing[a]

	BATCH	TIME-SHARING	DESKTOP	NETWORK
DECADE	1960s	1970s	1980s	1990s
TECHNOLOGY	MEDIUM-SCALE INTEGRATION	LARGE-SCALE INTEGRATION	VERY LARGE SCALE	ULTRA LARGE SCALE
LOCATION	COMPUTER ROOM	TERMINAL ROOM	DESKTOP	MOBILE
USERS	EXPERTS	SPECIALISTS	INDIVIDUALS	GROUPS
USER STATUS	SUBSERVIENCE	DEPENDENCE	INDEPENDENCE	FREEDOM
DATA	ALPHA-NUMERIC	TEXT, VECTOR	FONTS, GRAPHICS	SCRIPT, VOICE
OBJECTIVE	CALCULATE	ACCESS	PRESENT	COMMUNICATE
USER ACTIVITY	PUNCH & TRY (SUBMIT)	REMEMBER & TYPE (INTERACT)	SEE & POINT (DRIVE)	ASK & TELL (DELEGATE)
OPERATION	PROCESS	EDIT	LAYOUT	ORCHESTRATE
INTER-CONNECT	PERIPHERALS	TERMINALS	DESKTOPS	PALMTOPS
APPLICATIONS	CUSTOM	STANDARD	GENERIC	COMPONENTS
LANGUAGES	COBOL, FORTRAN	PL/1, BASIC	PASCAL, C	OBJECT ORIENTED

[a] From "Networked computing in the 1990s," by L. G. Tesler. Copyright © 1991 by Scientific American, Inc. All rights reserved.

throughout the network, and will communicate through the computers to other people.

A. Computer-Supported Cooperative Work

Computer-supported cooperative work (CSCW) is the term used to describe a group of people working together, with the aid of computers, on a common task. It can range from a group of people in a meeting room using laptop computers to contribute to a shared "electronic whiteboard" to a conference conducted over electronic mail. CSCW allows people in physically dispersed organizations to work together, and a study by Sproull and Kiesler (1991) showed that using a network induced the participants to talk more frankly and more equally. People who were reluctant to contribute to a face-to-face meeting reported feeling more lively and confident when expressing themselves in a computer conference.

However, computer-mediated communication is less rich than face-to-face discussion. It lacks the visual cues that signal, for example, impatience, annoyance, and dissent. Partly as a consequence, people in a computer conference took longer to reach a consensus than an equivalent face-to-face group, and they tended to express more extreme opinions. On a wider scale, communication by computer can alter patterns of work and the exercise of power within an organization. The free sharing of information and frank exchange of views that typifies network discussion may threaten established management and information channels within an organization. Worldwide networks such as the Internet allow an "information anarchy" across political boundaries.

The Internet already links together several million users on more than 300,000 computers in around 30 countries. An Internet user can send mail to any other user, connect to remote machines, and share work across the network. The World Wide Web gives a global hypertext network to each Internet user, so that a single set of options might offer satellite weather maps, catalogues from major academic libraries in the United States, and a program to search for the e-mail address of any user of the Internet. Clicking one of these options will automatically connect to the relevant computer and run software to display a map, show a library catalogue index, or return an e-mail address.

The information carried on Internet and the services it provides are not managed by a central agency, but are contributed by individual sites and users. Access to such a wealth and diversity of information can be seen as overwhelming (there are thousands of "newsgroups" on the Internet, each contributing a hundred or more messages a day), threatening (people throughout the world can exchange politically and socially inflammatory messages), or liberating (computer networks cut across established boundaries of power and influence). New social and technical challenges include finding ways to allow colleagues who have never met face to face to work together productively, developing "information rejection" methods to filter out unwanted information, and providing mechanisms to manage the electronic pathways, to stop them being choked by an exponential growth in communication or by deliberate sabotage using computer viruses. The phenomenal growth of the Internet offers new possibilities of global information exchange and brings new problems, of information overload, breaking of copyright, and lack of restraint.

B. Autonomous Agents

As networks grow beyond the comprehension of any individual, computers will play an increasing part in retrieving information and managing interaction. *Autonomous agents* are programs that assist with some computer-based task, such as arranging a meeting, finding a document across computer networks, or filtering e-mail according to priority. A user might ask an agent to arrange a meeting with selected colleagues around a given date. Once the agent is activated, it carries out the job of arranging a meeting autonomously. It consults the on-line calendars of the colleagues, finds a range of dates and times, suggests a suitable time and venue to the colleagues by mail, and when agreed, confirms the arrangement.

The style of interaction with an autonomous agent is very different from a command language such as DOS or the point and click of direct manipulation. Rather than issuing commands, the user specifies needs and constraints; for example, by partially completing a form or by indicating ranges on a scale. The interaction is one of request and negotiation. An autonomous agent requesting a meeting with the company director may need to ask and respond in a way very different from one that negotiates with a colleague. Computer etiquette and the micropolitics of organizations are new issues for software designers.

C. Ubiquitous Computing

In time, the computer may fade away altogether. An average house already has 10 or more microcomputers. They are embedded in washing machines, video recorders, telephones, remote controllers, thermostats, clocks, microwave ovens, door bells, cameras, toasters, and electric shavers. We do not notice them as computers because they are programmed to perform a restricted task and because their interface or system image is that of a familiar consumer device. However, they use general-purpose computer chips and in the future they will perform a wider range of tasks, such as providing diagnostic information, or communicating with other domestic appliances.

Ubiquitous computing can mean giving everyday objects computing power while retaining their familiar forms. A pad of paper might transmit what is written on it, a telephone might provide a printed transcript of a conversation, or a window might show an image of the scene outside some hours before. Or it can refer to the seamless integration of computer and noncomputer objects. Xerox EuroPARC are developing a DigitalDesk (Newman & Wellner, 1992) to demonstrate the movement of information between paper and computer. They have suspended a video camera and a projector above an ordinary desk, so that electronic documents can be projected onto the desk, and paper ones can automatically be digitized into computer text. Software connected to the camera will be able to recognize hand gestures, so that a person working at the desk can move the projected documents around just as they would push sheets of paper.

Perhaps the consumer device that will change the most as it is invaded by the computer is television. Digital television combines the interactivity of computer games with the image quality of television opening a vista of "edutainment" from interactive soap operas to self-guided tours of the great museums. It demands a new range of skills, combining television production, electronic art, and software design, and it will provide a new dimension to human–computer interaction, as millions of people simultaneously participate in an interactive television production.

D. The Challenge of Human–Computer Interaction

Human–computer interaction is a new discipline, and it has had little opportunity to mature, because computers and their users are changing so rapidly. At its worst, HCI is a mishmash of anecdotes and good intentions. But, at its best, it blends the psychology and technology of computing, begins to turn software design from an art into a science, and offers guidelines for good practice in developing and deploying computer systems. It has responded to new topics such as user-centered design and computer-supported cooperative working; and it has informed the design of exciting and usable computers such as the Apple Macintosh.

HCI must now address the wider social and organizational issues of living in a computer-mediated world. There are no straightforward technical answers to questions such as "what are the consequences of introducing computer networks into organizations?" "should different interfaces be developed for different cultures?" and "what tasks should never be replaced by computers?" The challenge of HCI is to show how computers can be used to empower people, to assist the design of computer systems that we can not only use with ease, but that we can also begin to respect and trust.

References

Apple Computer Inc. (1987). *Apple human interface guidelines: The Apple desktop interface.* Reading, MA: Adison-Wesley.

Baecker, R. M., & Buxton, W. A. S. (1987). An historical and intellectual perspective. In R. M. Baecker & W. A. S. Buxton (Eds.), *Readings in human-computer interaction* (pp. 41–54). San Mateo, CA: Morgan Kaufmann.

Bush, V. (1945). As we may think. *Atlantic Monthly, 76*(1), 101–108.

Card, S. K., Moran, T. P., & Newell, A. (1980). The keystroke-level model for user performance time with interactive systems. *Communications of the ACM, 23,* 396–410.

Card, S. K., Moran, T. P., & Newell, A. (1983). *The psychology of human-computer interaction.* Hillsdale, NJ: Erlbaum.

Diaper, D. (1989). Task Analysis for Knowledge Descriptions (TAKD); the method and an example. In D. Diaper (Ed.), *Task analysis for human-computer interaction* (pp. 108–159). Chichester: Ellis-Horwood.

Dix, A., Finlay, J., Abowd, G., & Beale, R. (1993). *Human-computer interaction.* New York: Prentice-Hall.

Engelbart, D. C., & English, W. K. (1988). A research center for augmenting human intellect. In I. Greif (Ed.), *Computer-supported cooperative work: A book of readings* (pp. 81–105). Palo Alto, CA: Morgan Kaufmann.

Fitts, P. M. (1954). The information capacity of the human motor system in controlling amplitude of movement. *Journal of Experimental Psychology, 47,* 381–391.

Flower, L. S., & Hayes, J. R. (1980). The dynamics of composing: Making plans and juggling constraints. In L. Gregg & E. Steinberg (Eds.), *Cognitive processes in writing: An interdisciplinary approach* (pp. 31–49). Hillsdale, NJ: Erlbaum.

Harel, D. (1988). On visual formalisms. *Communications of the ACM, 31*(5), 514–530.

Hewitt, B., Gilbert, N., Jirotka, M., & Wilbur, S. (1990). *Theories of multi-party interaction* (Technical Report). London: Social and Computer Sciences Research Group, University of Surrey and Queen Mary and Westfield Colleges, University of London.

Hutchins, E. (1990). The technology of team navigation. In J. Galegher, R. E. Kraut, & C. Edigo (Eds.), *Intellectual teamwork* (pp. 191–322). Hillsdale, NJ: Erlbaum.

Jeffries, R., Miller, J. R., Wharton, C., & Uyeda, K. M. (1991). User interface evaluation in the real world: A comparison of four techniques. In *Proceedings of ACM CHI '91* New Orleans, LA (pp. 119–124). New York: ACM Press.

Jeffries, R., Turner, A. A., Polson, P. G., & Atwood, M. E. (1981). The processes involved in designing software. In J. R. Anderson (Eds.), *Cognitive skills and their acquisition* (pp. 255–283). Hillsdale, NJ: Erlbaum.

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

Landauer, T. K. (1987). Relations between cognitive psychology and computer system design. In J. M. Carroll (Ed.), *Interfacing thought: Cognitive aspects of human–computer interaction* (pp. 1–25). Cambridge, MA: MIT Press.

Maguire, M. C. (1990). A review of human factors guidelines and techniques for the design of graphical human-computer interfaces. In J. Preece & L. Keller (Eds.), *Human-computer interaction* (pp. 161–184). Hemel Hempstead: Prentice-Hall International.

Malone, T. W. (1981, December). What makes computer games fun? *BYTE,* pp. 258–277.

Michie, D., & Johnston, R. (1984). *The creative computer: Machine intelligence and human knowledge.* Harmondsworth: Penguin.

Miller, G. A. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review, 63,* 81–97.

Newell, A., & Simon, H. (1972). *Human problem solving.* Englewood Cliffs, NJ: Prentice-Hall.

Newman, W. M., & Wellner, P. (1992). *A desk supporting computer-based interaction with paper documents* (Tech. Rep. EPC-91-131). Cambridge, UK: Rank Xerox EuroPARC.

Norman, D. A. (1986). Cognitive engineering. In D. A. Norman & S. W. Draper (Eds.), *User centered system design* (pp. 31–61). Hillsdale, NJ: Erlbaum.

Papert, S. (1980). *Mindstorms: Children, computers and powerful ideas.* New York: Basic Books.

Reece, J. (1993). *Cognitive processes in the development of written composition skills: The role of planning, dictation and computer tools.* Doctoral thesis, La Trobe University, Melbourne, Australia.

Sharples, M. (1993). A study of breakdowns and repairs in a computer-mediated communication system. *Interacting with Computers, 5*(1), 61–77.

Shneiderman, B. (1992). *Designing the user interface: Strategies for effective human-computer interaction.* Reading, MA: Addison-Wesley.

Spivey, J. M. (1988). *The Z notation: A reference manual.* Hemel Hempstead: Prentice-Hall International.

Sproull, L., & Kiesler, S. (1991, September). Computers, networks and work. *Scientific American, 265*(3), 84–91.

Tesler, L. G. (1991, September). Networked computing in the 1990s. *Scientific American, 265*(3), 54–61.

The Open University (1990). *A guide to usability.* Milton Keynes: The Open University.

Young, R. M. (1983). Surrogates and mappings: Two kinds of conceptual models for interactive devices. In D. Gentner & A. L. Stevens (Eds.), *Mental models* (pp. 35–52). Hillsdale, NJ: Erlbaum.