HUMAN-COMPUTER INTERACTION: Psychology as a Science of Design

Human-computer interaction (HCI) study is the region of intersection between psychology and the social sciences, on the one hand, and computer science and technology, on the other. HCI researchers analyze and design specific user interface technologies (e.g. pointing devices). They study and improve the processes of technology development (e.g. task analysis, design rationale). They develop and evaluate new applications of technology (e.g. word processors, digital libraries). Throughout the past two decades, HCI has progressively integrated its scientific concerns with the engineering goal of improving the usability of computer systems and applications, which has resulted in a body of technical knowledge and methodology. HCI continues to provide a challenging test domain for applying and developing psychological and social theory in the context of technology development and use.

THE EMERGENCE OF USABILITY Human-computer interaction (HCI)

Has emerged relatively recently as a highly successful area of computer science research and development and of applied psychology. Some of the reasons for this success are clearly technical. HCI has evoked many difficult problems and elegant solutions in the recent history of computing, e.g. in work on direct manipulation interfaces, user interface management systems, task-oriented help and instruction, and computer-supported collaborative work. Other reasons for its success are broadly cultural: The province of HCI is the view the nonspecialist public has of computer and information technology and its impact on their lives; HCI is the visible part of computer science. The most recent reasons for the success of HCI are commercial: As the underlying technologies of computing become commodities, inscribed on generic chips, the noncommodity value of computer products resides in applications and user interfaces, that is, in HCI. HCI has evolved rapidly in the past two decades as it has struggled to develop a scientific basis and utility in system and software development. In this chapter, I review the progression of HCI toward a science of design. My touchstone is Simon's (1969) provocative book The Sciences of the Artificial. The book entirely predates HCI, and many of its specific characterizations and claims about design are no longer authoritative (see Ehn 1988). Nevertheless, two of Simon's themes echo through the history of HCI and still provide guidance in charting its continuing development. Early in the book, Simon discussed the apparently complex path of an ant traversing a beach, observing that the structure of the ant's behavior derives chiefly from the beach; the ant pursues a relatively simple goal and accommodates to whatever the beach presents. The external world, including technology human beings create, should be expected to play a powerful role in structuring human behavior and experience. Late in the book, Simon sounds the second theme: the need for a science of design, a research paradigm and university curriculum directed at understanding, furthering, and disseminating design knowledge. He lamented the tendency of engineering disciplines to adopt goals and methodologies from the natural sciences, to their detriment with respect to design. HCI is a science of design. It seeks to understand and support human beings interacting with and through technology. Much of the structure of this interaction derives from the technology, and

many of the interventions must be made through the design of technology. HCI is not merely applied psychology. It has guided and developed the basic science as much as it has taken direction from it. It illustrates possibilities of psychology as a design science.

Software Psychology

The work that constitutes the historical foundation of HCI was called "software psychology" in the 1970s (e.g. Shneiderman 1980). The goal then was to establish the utility of a behavioral approach to understanding software design, programming, and the use of interactive systems, and to motivate and guide system developers to consider the characteristics of human beings. Software psychology had two distinct methodological axioms. The first assumed the validity of a received view of system and software development, namely the so-called waterfall model of top-down decomposition and discretely sequenced stages with well-specified hand-offs (e.g. Royce 1970). The second assumed two central roles for psychology within this context: (a) to produce general descriptions of human beings interacting with systems and software, which could be synthesized as guidelines for developers; and (b) to directly verify the usability of systems and software as (or more typically, after) they were developed. Software psychology inaugurated a variety of technical projects pertaining to what we now call the usability of systems and software: assessing the relative complexity of syntactic constructions in programming languages (e.g. Sime et al 1973), classifying people's errors in specifying queries and procedures (Miller 1974), describing the utility of mnemonic variable names and in-line program comments (Weissman 1974), and explicating how flowcharts serve as a programming aid (Shneiderman et al 1977). This work inspired many industrial human-factors groups to expand the scope of their responsibilities to include support for programming groups and the usability of software. The basic axioms of software psychology proved to be problematic. The waterfall idealization of design work is infeasible and ineffective (e.g. Brooks 1975/1995). It is only observed when enforced, and it is best regarded as a crude management tool for very large-scale, long-term projects. As computer research and development diversified in the 1970s and 1980s, small and distributed personal work organizations became more commonplace. Product development cycles were often compressed to less than a year. The two roles assigned to software psychologists were also problematic and resulted in a division of labor. Researchers, mainly in universities, developed general descriptions of users and framed them as general guidelines. Humanfactors specialists in industry tried to apply these guidelines in specific projects. This division did not work particularly well. From the standpoint of practical goals, the research of this period tended to focus on unrepresentative situations (e.g. undergraduates standing in for programmers, 50-line programs standing in for business systems, and teletypes standing in for display tubes). To obtain statistically stable results, researchers often created outrageous contrasts (organized versus disorganized menus, structured versus scrambled programs). The researchers sometimes understood little about the users of their guidelines and proffered superfluous advice.

Psychologists and others playing the human-factors specialist role in industry were frustrated trying to use and encourage use of these guidelines. They were also frustrated in their other role of verifying the usability of finished systems, because the research tools they had (formal experiments aimed at specific differences among alternatives) were too costly and too

uninformative to allow them to serve as anything more than gatekeepers. Human-factors specialists were often seen as imposing bureaucratic obstacles blocking heroic developers. The origins of HCI in software psychology posed two central problems for the field in the 1980s. One problem was to better describe design and development work and to understand how it could be supported. The other problem was to better specify the role that psychology, in particular, and social and behavioral science, more broadly, should play in HCI.

Iterative Development

Starting in the 1970s, empirical studies of the design process began to explicate the difficulties of the waterfall model. Design work is frequently piecemeal, concrete, and iterative. Designers may work on a single requirement at a time, embody it in a scenario of user interaction to understand it, reason about and develop a partial solution to address it, and then test the partial solution—all quite tentatively—before moving on to consider other requirements. During this process, they sometimes radically reformulate the fundamental goals and constraints of the problem. Rather than a chaos, it is a highly involuted, highly structured process of problem discovery and clarification in the context of unbounded complexity (e.g. Carroll et al 1979, Curtis et al 1988, Malhotra et al 1980).

The leading idea is that designers often need to do design to adequately understand design problems. One prominent empirical case was Brooks's (1975/1995) analysis of the development of the IBM 360 Operating System, one of the largest and most scrupulously planned software design projects of its era. Brooks, the project manager, concluded that system and software designers should always "plan to throw one away" (pp. 116–23). This was a striking lesson to draw and carried with it many implications. For example, formal and comprehensive planning and specification aids (such as detailed flowcharts) will have limited utility in supporting such an iterative design process.

This reformulation of design created an opening for new ideas. Noteworthy inspiration came from the work of the great industrial designer Henry Dreyfuss, who had pioneered an empirical approach in the 1940s (Dreyfuss 1955).

Dreyfuss's approach institutionalizes an accommodation to designers' propensity for concrete, incremental reasoning and testing. It incorporates four central ideas: (a) early prototyping with (b) the involvement of real users, (c) introduction of new functions through familiar "survival forms," and (d) many cycles of design iteration. Dreyfuss pushed beyond the designer's need for prototyping and iteration as a means of clarifying the design problem (also emphasized by Brooks 1975/1995) to the user's knowledge, experience, and involvement to constrain design solutions.

A typical example is Dreyfuss's design work on airplane interiors for Lockheed. Dreyfuss sent two associates back and forth across the United States on commercial flights to inventory passenger experiences. They found that passengers were often baffled by design details like water taps that were unnecessarily novel. They were impressed that people wanted to think of airplane seats as armchairs and not as "devices." Initial designs were prototyped in a Manhattan warehouse, and a full flight of "passengers" was hired to occupy the mock-up for 10 hours: to store carry-on luggage,

to eat meals, to use lavatories. These tests concretized requirements for seating, storage space, lavatory-door latches, and so forth, and permitted low-cost, iterative reworking of the original designs.

In the 1980s, the inevitability of an empirical orientation toward system and software design rapidly evolved from a somewhat revolutionary perspective to the establishment view. This development provided early and critical motivation and direction for research on user interface management systems to enable prototyping (Tanner & Buxton 1985), it encouraged user participation in design (Gould & Boies 1983, Nygaard 1979, Pava 1983), it emphasized user interface metaphors for presenting novel functionality through familiar concepts (Carroll & Thomas 1982, Smith et al 1982), and it made “rapid prototyping” a standard system development methodology (Wasserman & Shewmake 1982, 1985).

Iterative development shifted the focus of usability evaluation from summative to formative (Scriven 1967). Formal experiments are fine for determining which of two designs are better on a set of a priori dimensions, but they are neither flexible nor rich enough to guide continual redesign. “Thinking aloud” had been pioneered by deGroot (1965) and Newell & Simon (1972) in the study of puzzles and games, and it had been shown to vividly illuminate strategic thought (Ericsson & Simon 1985). In the 1980s, thinking aloud became the central empirical, formative evaluation method in HCI (e.g. Mack et al 1983, Wright & Converse 1992).