4
Comparative Task Analysis: An Alternative Direction for Human–Computer Interaction Science

Ruven Brooks

What Would Designers Like from Human–Computer Interaction Science?

Consider the following situation. A human–computer interaction specialist is approached by the head of a software design project with the following request:

> We have some people who perform problem-solving task F. They are highly skilled at this task and have spent a long time learning to do F well. Since F is so important to our organization, our management is willing to devote considerable resources to developing new, more interactive tools for doing F. We'd like you to analyze the behavior of these people and tell us what kinds of tools we should build.

If I am the HCI specialist and if I have options as to how to respond to the request, I might begin by wondering what kind of response the project manager would find most useful. If the manager does, in fact, have the task of designing the tools completely, as versus providing a new interface to existing tools, then the following example of analysis might fulfill the manager's expectations:

> When F is described using the N notation, problem solving in F seems to occur in two distinct phases. The first phase consists of two activities, and there is a great deal of alternation between them. Both activities 1 and 2 seem to be class C_{27} tasks that are characterized by high memory loads. As the task is now structured, workers must remember information retrieved in activity 1 throughout the processing of an M in activity 2. Since activity 2 is also a high-memory-load task, they have difficulty in remembering the activity 1 information, and they frequently need to return to activity 1 to retrieve the same information. Based on what has proved useful in other applications such as A_{27} and A_{59} with similar structure, what is needed is an external memory tool, in the form of a P, S, or Q display that shows the following six types of information for each M: . . .
>
> The second phase of F is basically a C_{18} task but with aspect α_{κ} reduced to only a brief notation. The focus of activity shifts to performing the K analysis on each M that has been processed. Experience with other C_{18} tasks suggests that it is useful to look at the information generated in the K analysis to see whether it can be broken apart into independent categories. In fact, the information produced by K analysis can be divided into four types, which the current K analysis tool generates all in one pass and which requires a large number of input parameters, all of which must be correct for the program to succeed. According to the K-analysis specialist, the four types of information are calculated independently, but some of the inputs overlap. Based on the experience with the A_{52} application, which is also a C_{18} task, reorganizing the K analysis into four separate programs is likely to reduce errors despite the redundant entry of parameters.

This sort of response has three properties that are likely to be particularly valuable to the manager in designing the system. First, the response is based on the use

of systematically recorded knowledge about human behavior. Although there are clearly insightful individuals in every profession or occupation who are able to observe human behavior objectively, most specialists in domains other than psychology are unlikely to have the skill or knowledge to relate their observations to observations made in other domains. The contribution of the HCI specialist is distinct and, therefore, valuable.

Second, the response is presented in a framework and at a level of analysis that is useful for evaluating the impact of the major design decisions on user behavior. Dividing user activities into phases based on the kinds of problems being solved or on the kinds of information being used for decision making provides a useful basis for both partitioning functionality across different software tools and for predicting possible impacts of new tools on the problem-solving process. An analysis, such as traditional task analysis, that focused only on the individual steps in performing F would not be as useful for partitioning functionality.

Third, the response is highly specific to activity F. It suggests an actual design for the programs and interfaces that should be built for doing F, rather than suggesting principles or guidelines to be used in the design of such tools or metrics for evaluating how good the tools are.

HCI as Descriptive, Engineering Science

As useful as this form of response might be to the designer, most HCI specialists, particularly those trained as psychologists, would feel uncomfortable with producing it. There would probably be two primary bases for their discomfort. First, the advice being given was not based on a model of problem-solving behavior, at least not in the "technical theory" sense used by Newell and Card (1985). Instead, the basis of the analysis is classification of the new task in some kind of task taxonomy and the instantiation of schemata from that taxonomy. There is no reduction of the task to a combination of fundamental, idealized behavioral mechanisms or modeling elements, nor is there any claim that the taxonomy can be derived from a more general model of human behavior.

A second major problem is that the recommendations take the HCI specialist out of the ancillary role of providing tools or methodologies for interface design and into the primary role of the creation of the design itself. To fill this role, the computer-interaction specialist must have two kinds of knowledge:

Knowledge about F, perhaps as much as someone whose main occupation is F, and must be able to represent and describe F explicitly at a very detailed level;

Knowledge about the specifics of other kinds of tasks and about the interface designs that have been used for them (it may even be necessary for the specialist to understand the limitations and capabilities of existing tools for implementing the interface).

Granting for the moment the usefulness of the kind of response given to the software designer, what kind of discipline could produce it? That is, what sorts of

models and methodologies should the field of HCI develop if it wants to produce this kind of analysis and advice from a sound scientific or engineering basis, so that the HCI specialists will be competent to provide it?

Adequate Description as Prerequisite to Technical Theory

Newell and Card (1985) argue that what HCI science ought to focus on developing is "technical" theories of HCI. By a technical theory, they mean one that is formal, manipulation-oriented, idealization-based, and cumulative. They argue that such theories provide a sound basis for design because they can provide both abstractions that encourage consideration of a broader or different range of design possibilities and design aids, such as tools for quantitatively evaluating alternatives. As examples, they cite organic chemistry, solid-state physics, molecular genetics, and mechanical engineering.

The previous view of HCI as relying heavily on description and taxonomy seems to contrast strongly with this position. In fact, an examination of some aspects of the earlier position suggests such a descriptive and taxonomic discipline is a natural and probably necessary part of evolution to such theoretical science. A starting point concerns the question of how technical theories for a design discipline are generated. Do they arise from reduction and specialization of some broader, more basic science or are they derived independently? One model of the relationship is that the natural science, physics, uncovers or proposes great laws or equations that describe enduring properties of the physical world. It does this by making hypotheses and collecting data to test the hypotheses. The resultant theories, however, may contain unknowns that are unmeasurable or forms that are insoluble. For example, Maxwell's Laws describe the fundamental electromagnetic interactions, but the differential equations involved do not have general closed-form solutions. To convert these general laws into practical engineering analysis or design tools, engineering makes appropriate simplifying assumptions that allow available numerical techniques to be applied. Assumptions about boundary conditions allow Maxwell's Laws to be simplified so that finite element analysis can be applied.

In general, though, the initial derivation of engineering science from basic science is an extremely rare event. Instead, what typically happens is that engineering analyses are invented independently of basic science. For example, circuit theory describes a set of phenomena that could be equally well described with Maxwell's equations, but which were invented independently because, until recently, Maxwell's equations could not be applied directly. In many engineering disciplines, such as civil engineering, most of the useful models never have been related back to physics or chemistry. Models in civil engineering for runoff and groundwater levels, for example, are not directly relatable to models of fluid flow in porous media.

Even if it is conceptually possible to relate an independently derived engineering theory back to a basic science, there may be no engineering motivation for doing so. In civil engineering, for example, much of the problem with existing models is in obtaining sufficient, meaningful measurements to apply them. Relating them to a more general formulation with even more unknowns is not a particularly attractive activity.

How then do engineering theories arise? A necessary kernel is the development of an appropriate abstraction that discards irrelevant details while isolating and emphasizing those properties of artifacts and situations that are most significant for design. Indeed, this property of abstraction may be more important than the extent to which the abstraction gives rise to manipulatable formalisms or prediction; by indicating what properties of an artifact really are significant, a good abstraction may lead both to invention of new artifacts that produce these aspects in novel ways and to novel uses for existing artifacts.

Given this characteristic, how do useful abstractions arise? Rarely will they be imported, isolated from the rest of their theoretical structure, from some more basic science; instead, it is far more likely that they will be derived from first-order descriptions or taxonomies for the artifacts and phenomena of the design discipline. The most probable reason why structural engineering views arbitrary structures in terms of frames and trusses is because this is the way actual structural components were described at the time the analyses were being developed; in a world in which all architecture consisted of domes, structural engineering would probably have different analysis primitives.

Newell and Card see the role of description as primarily that of identifying the phenomena of interest in a field: "Purely descriptive studies . . . can make important contributions to HCI, if they describe something novel or describe a previously unnoticed aspect of user behavior or serve as a major confirmation of other descriptive studies." In contrast, from the perspective taken here, developing good descriptions, particularly good taxonomy that adequately differentiates artifacts, is important in its own right because it is a useful, possibly necessary, precursor to appropriate formalism and, in turn, to technical theory.

Description as a Means of Communicating Design Experience

In addition to the role that artifact and phenomena description plays in giving rise to abstraction and theory formalization, it continues to play a central role in recording and communicating design experience, even in the mature stages of a design science when more formal analyses are available. The reason that this is frequently the case is that formal models or technical theories try to describe artifacts in as explicit detail as possible; indeed, much of their power depends on having such detailed information available. To make use of circuit theory, one must specify the interconnections and values of each individual component in the circuit. For all but the simplest circuits, though, it will be very difficult for an engineer to decide on the basis of the circuit diagram or, even, the diagram plus input and output signals, what the circuit does and whether it might be useful in some future design.

In contrast, describing a circuit as a "delta modulator analog to digital converter" enables the electrical engineer to recognize the circuit design as one that can be used in other contexts, although it may be little direct help in determining the electrical properties. (It is worth noting that there are many cases in which the

mapping does go in both directions; describing a circuit as a "4th order Butterworth filter" both enables an engineer to recognize what the circuit does and provides a starting point for signal analysis.)

Good descriptive languages or systems must be able to express both the requirements or situations that gave rise to particular designs and the structure and properties of the designs themselves. In some engineering disciplines, such as electrical engineering or mechanical engineering, design requirements frequently grow out of the characteristics of other engineered artifacts whose properties are known; in these disciplines, the descriptive language focuses only on the artifacts that are being created, with the requirements being incorporated into the description of the artifacts. Referring to a device as a "delta modulator analog to digital converter" both describes the structure of the device as using delta modulation and gives its function, analog to digital conversion.

In other disciplines, where design requirements are derived from naturally occurring situations, accurate determination and description of the requirements become problems in their own right. In these disciplines, descriptive systems arise to characterize both different classes of design problems and different types of solutions. For example, in civil engineering, a standard textbook on earth-dam design (Sherard, Woodward, Gizienski, & Clevenger, 1963) has chapters both on the environment in which the dam will be built, such as foundation-soil type, earthquake hazard, and wave severity, and on dam designs and construction techniques, such as the use of thin-core versus homogeneous construction.

Making these distinctions serves a purpose roughly analogous to the diagnosis–treatment separation in medicine. Although the mapping from requirements to design may be one to one and "pathognomonic" (distinctly characteristic of a particular disease), keeping them separate at the descriptive level helps to ensure that the requirements are adequately characterized before design takes place. In the case where multiple designs are viable for a given set of requirements, it provides a framework in which the consideration of alternatives is encouraged.

The Current Status of Description in HCI Science

The preceding section argues for the importance of good descriptive systems in engineering or design sciences, both as a prerequisite to developing technical scientific theories and as a necessary prerequisite to accumulating design experience. As a relatively new design science, it is important, for both of these reasons, that HCI have powerful descriptive capabilities available. HCI most closely resembles the class of engineering disciplines, such as civil engineering and petroleum engineering, in which most of the requirements for designs arise out of naturally occurring entities or environments whose properties require considerable investigation before design can begin. In the case of HCI, the naturally occurring entities or environments are those of human behavior, both in isolated cognition and in social interaction.

Following the model for description used in these areas, the necessary descriptive capabilities for HCI should probably be of two distinct types. Languages or notations for describing tasks should permit describing the tasks for which computer

applications and computer interfaces are being developed in such a way that commonalities and differences among the tasks are revealed. Similarly, languages or notations for describing computer systems from the standpoint of their appearance to their users should make it possible to identify broad classes of application and interface designs while, at the same time, permitting comparison of similar designs.

Describing Tasks

The description of tasks is an area in which the basic, cognitive science aspects of HCI potentially have the most to contribute. Certainly, HCI science and its broader predecessor, human-factors engineering, do not lack for descriptive and analytic notations; the question is as to whether they are at the right level and have the right sort of architecture to support matching to candidate designs.

One obvious candidate is the technique called *task analysis* from human-factors engineering (Drury, Paramore, Van Cott, Grey, & Corlett, 1987). This technique involves breaking down tasks into components and specifying the flow between components. In order to insure that all of the behavior is captured, it is important that all of the components be specified at as atomic a level as possible.

As the starting point for an HCI task-analysis language, human-factors task analysis has major drawbacks. First, the nature of the components is currently open and ad hoc; a typical example of an atomic task for a nuclear-power-plant monitoring system is "start recirculation pumps." Although there has been some effort to restrict the components by the use of "vocabulary control" for describing them, there is no widely accepted primitive set. This makes cross-task comparison difficult. Does "start recirculation pumps" involve just flipping a switch or does it require adjusting a control until a meter reaches a certain level?

Even if such a set of primitives became widely accepted, though, the very low level of the primitives and the lack of any formalism for summarization makes it difficult to use human-factors task analysis as the basis for task description in HCI. Suppose that such a task analysis were carried out for controlling a nuclear power plant and for piloting an aircraft. Each analysis might have thousands of instances of the task-analysis primitives. How could the similarities and differences between the tasks be determined?

Most of the more recent descriptive efforts in HCI science have focused on developing user models that can be used to analyze, predict, or explain the performance of users with different interface and system designs. Do user modeling systems provide the necessary descriptive capabilities?

The GOMS model (Card, Moran, & Newell, 1983) with its production system representation (Polson & Kieras, 1985) is probably the best established HCI user model, at least in terms of the body of supporting research, and is illustrative of what can be done with user models. Because the intent of the GOMS model is to capture differences in user behavior for specific tasks, its constructs – goals, operators, methods, and selection – are not primitives in the sense of a ground alphabet of symbols out of which higher level constructs are assembled; there is no exhaustive list of, say, goal primitives out of which all goals for all users can be constructed.

Rather, they specify classes that need to be filled in order to construct user models for the specific behaviors and activities. For example, to construct a GOMS model of the manuscript-editing task, one supplies task-specific goals such as EDIT-MANUSCRIPT or LOCATE-LINE.

Constructing a GOMS model does involve task analysis to identify the operations and basic objects of the task that will be manipulated by the methods, but, again, the constructs used for this purpose are ad hoc. In the manuscript-editing task, the constructs include WORD, TEXT-SEGMENT, and PLACE-IN-MANUSCRIPT. An entirely different set of constructs would presumably be needed for, say, task analysis for statistical data analysis.

Although models such as GOMS are of value in comparing interface or application designs within tasks, they have the same problems as human-factors task analysis for deciding what the similarities or differences are between tasks. Consider the problem of deciding whether a hydrocarbon well log data interpretation system should have the same user interface as a statistical data analysis system. Even if a complete GOMS unit-task analysis were available for the statistical analysis system interface, there is no way to decide how to compare the unit tasks for statistical analysis with those for hydrocarbon well log interpretation. (In fairness to the authors of the GOMS model, it must be pointed out that they do not advocate GOMS analysis for task description and note the lack of cognitive skills taxonomy. In its absence, they do provide an informal chart listing some possible skill dimensions and comparing a number of tasks on them, but they do not consider this to be even the beginnings of a powerful system.)

Are there other candidates? Developmental psychology and psychometrics may have something to offer, but the best current bets may come from outside of HCI science or psychological science, from the discipline of "knowledge engineering" for the construction of expert systems. Initially, work in this area attempted to use a single universal set of methods and techniques for all applications. More recently, though, attention has focused on developing problem-solving methods and systems that are specialized for particular classes of problems (Marcus, 1988), and McDermott (1988) has proposed a taxonomy of these methods. An example of a method within that taxonomy is *propose-and-revise,* which can be used to solve design problems that involve selecting a configuration of components from a fixed catalog.

Although not made explicit, this method taxonomy is presumably equivalent to a taxonomy of tasks that could be attacked by these methods. How to go about classifying a new task is also unaddressed, but using a taxonomy based on the methods required to solve problems seems to hold promise that once a task has been analyzed using this taxonomy, it will be relatively easy both to compare it to other tasks and to map from the task onto computer application and interface designs.

Describing User Interfaces

Strangely enough, given the central role of notation in computer science, a similar description deficiency exists on the interface side. There is, of course, a large and

rapidly growing set of formalisms used to describe user interfaces that range from tool kits of components for actual interface construction to various kinds of state diagrams for specifying flow of control to algebraic specification of the semantics of interface components. There are two problems with these notations. The first is that most of these specifications are quite detailed and low-leveled. In some of them, such as the X Window tool kits, the position at which a menu pops up is specified in terms of screen pixel coordinates. Comparing two different interfaces to say how they are alike or different can be done only at the level of line-by-line comparison of specification code, and there are no ways to characterize more generally the differences that are found other than by listing them exhaustively. This makes it impossible to characterize an interface style or "look and feel" except, perhaps, by vague comparison to some existing, well-known interface. Were someone to publish a catalog of interface designs based on this level of description, it is hard to conceive of what the table of contents or the index would look like.

The second problem with using software specification languages as interface description languages for HCI is that the linkage to the task is often obscure. At best, those pieces of software built using facilities structured on a user-interface management system model (as versus a tool kit model) clearly separate the specification of the interface from the calls to the underlying application routines, but it does not make clear the application structure. The notation might make clear that, when "CompleteScoring" is selected from a menu, appl-comp-scr will be invoked, but it provides no help in understanding what the "appl-comp-scr" routine does.

At least the first of these problems is currently under attack from within the user-interface design and development community. There is considerable work underway directed toward higher-level description of interface designs and characteristics. For example, Apperley and Spence (1989) present a notation for menu structures that can specify behavioral characteristics of menus such as whether they are multichoice or single choice and whether they invoke submenus. A description of the entire menu structure of an application of moderate complexity could fit on a single page when shown in a diagrammatic form. An equivalent text string notation has potential as a basis for comparison of applications.

Another example of work in this area, task-action grammars (Payne & Green, 1989), is still basically intended to map between user-interface actions and implementation software routines but even offers some possibilities for addressing the second problem. The "Task Features" and "Co-Occurrence Restrictions" constructs in the grammar can be used to express information about the task beyond the routines of the software implementation. While comparison of simple interfaces by comparing their task-action grammars has been demonstrated, the extension to more realistic interfaces remains an open question.

Directions for Description in HCI

Work on comparative description of interfaces is, in part, driven by the needs of software construction and is therefore likely to progress, although perhaps not in the directions most serviceable to HCI. Work on comparative description of tasks is also

likely to go forward, driven by the knowledge-engineering community. What kinds of description system development are still a worthwhile focus for HCI activity?

As the work on problem-solving methods for expert systems illustrates, there may be a large variety of bases for defining task taxonomies, and the taxonomies can vary widely in the range of tasks they cover. The focus of the knowledge-engineering community is likely to be on taxonomies for those aspects of tasks for which automated problem-solving methods are available, and taxonomy refinement and differentiation is likely to be driven more by the appearance of new problem-solving methods than by desires for more detailed or more accurate task description. Furthermore, they are unlikely to provide much for tasks for which they see the problem-solving methods as uninteresting; a taxonomy of data entry tasks is unlikely to emerge from work on inference techniques. Although the knowledge-engineering community is likely to be a source of some types of task taxonomies, they are unlikely to provide the full range of description that would be useful to designers, and further work on description and taxonomy of tasks from an HCI perspective is still worthwhile.

This work ought to have three properties. First, description systems should be hierarchical in the sense that a given description should start with major descriptive categories or labels and then provide further detail within each of these categories. For example, a general class of "qualitative model-fitting tasks" might be defined in which the computing system was being used as an aid to fit a model to empirical data. This might be further broken down into tasks for which the primary problem was to select the mathematical form of the model versus those in which the form of the model is known, but in which parameters must be manually selected and adjusted for an optimal fit. Descriptions that have this structure lend themselves to differential comparison more readily than formalisms, such as those used for task analysis, that are essentially flat with only one level of detail.

Second, descriptive distinctions should have associated operational tests or procedures that can be used to perform classification on new tasks. At this stage, informal but operational distinctions should be preferred over theoretically rigorous but nonoperational ones. For example, suppose that a category of "generate simulation from a theoretical model" were proposed to complement the "qualitative model-fitting" classification given above. Both categories involve working with sets of numerical data, but an associated, operational distinction that could be used to separate tasks was whether an initial set of empirical data was available. Following the argument already given for description as a precondition for technical theory, starting with operational categories is likely to produce analyses and theory that are directly applicable to the actual artifacts and constructs available to designers.

Finally, and most important, the task descriptions should capture the commonalities and differences among the tasks in regard to their interface requirements. For example, social science data analysis, some kinds of economic forecasting, and hydrocarbon well log interpretation might all be described as "iterative parameter fitting of qualitative models to a set of empirical data." Describing all of these tasks as "iterative parameter fitting" suggests that all of them may require interface facilities for keeping track of the multiple versions of the set of parameter values. In contrast, if they were all "model selection" tasks, the interface would have to support tracking the different forms of model that had been tried, instead of the values of the parameters.

Readers of professional publications from a range of engineering disciplines are familiar with the calls for making these disciplines more scientific and rigorous by providing models and analytical tools with more formal structure and derivations from underlying scientific bases. Certainly, there is a similar need in HCI, and, as HCI evolves, more such models and theories ought to appear. In the interim, further work on description along these lines will be directly usable in design and may even be a necessary precondition to the emergence of theories.

Acknowledgments

The comments of other participants in the work were extremely valuable; in particular, Mary Beth Rosson's comments forced me to make explicit distinctions about types of description, which were vague in the original draft.

References

Apperly, M. D., & Spence, R. (1989). Lean cuisine: A low-fat notation for menus. *Interacting with Computers, 1*(1), 43–68.

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

Drury, C. G., Paramore, B., Van Cott, H. P., Grey, S. M., & Corlett, E. N. (1987). Task analysis. In G. Salvendy (Ed.), *Handbook of human factors* (pp. 370–401). New York: Wiley.

Marcus, S. (1988). *Automating knowledge acquisition for expert systems*. Boston: Kluwer Academic Publishers.

McDermott, J. (1988). Preliminary steps toward a taxonomy of problem-solving methods. In S. Marcus (Ed.), *Automating knowledge acquisition for expert systems* (pp. 225–256). Boston: Kluwer Academic Publishers.

Newell, A., & Card, S. K. (1985). The prospects for psychological science in human–computer interaction. *Human–Computer Interaction, 1*(3), 209–242.

Payne, S. J., & Green, T. R. G. (1989). The structure of command languages: An experiment on task–action grammar. *International Journal of Man–Machine Studies, 30*(2), 194–213.

Polson, P. G., & Kieras, D. E. (1985). A quantitative model of the learning and performance of text editing knowledge. In L. Borman & B. Curtis (Eds.), *Proceedings of CHI '85: Human Factors in Computing Systems* (pp. 207–212). New York: ACM.

Sherard, J. L., Woodward, R. J., Gizienski, S. F., & Clevenger, W. A. (1963). *Earth and earth-rock dams*. New York: Wiley.