Buxton, W. & Sniderman, R. (1980). Iteration in the Design of the Human-Computer Interface. Proceedings of the 13th Annual Meeting, Human Factors Association of Canada, 72-81.

ITERATION IN THE DESIGN OF THE HUMAN-COMPUTER INTERFACE1

William Buxton and Richard Sniderman

Structured Sound Synthesis Project (SSSP)
Computer Systems Research Group
University of Toronto
Toronto, Ontario

ABSTRACT

Issues pertaining to designing effective human-computer interfaces are discussed. This presentation focuses on the special case of providing congenial computer-based tools to end users who are expert in their own area, but who may be technologically naive. In so doing, we draw examples from a particular study of designing computer systems for professional musicians. This experience brings to light many issues which have relevance beyond the specific application of music. These include the importance of effective prototyping tools, the use of test subjects during the design process, the importance of developing methods of performance evaluation, and more generally, the value of taking an iterative approach to design.

1. Introduction

Recent advances in the microelectronic industry have made wide-spread physical and economic access to high technology tools a reality. Before the full potential benefits of these tools can be realized , however, them must become accessible in the cognitive sense as well. This we can discuss from two points of view: that of the professional programmer, who must be able to structure complex programs. Using this technology, and that of the end user, who while being quite advanced in his own application area, is usually a layman in computer technology. Each case is an area of study in Human Factors with its own special properties, given the two groups differing abilities to deal with technologically-based problems.

In this paper, we are mainly concerned with tie problems of the end user. In the discussion, we follow the premise that if technologically-based tools are adopted , it is because the scope or magnitude of the user's problems have outgrown current techniques for dealing with them. Adopting the new tool must, therefore, eliminate problems, rather than create additional, or alternative ones.

The consequence of this premise is the specification that the new tool must adapt to the user, rather than force the user to adapt to it (which is too often the case today). By this, we mean that the system must "fit" the end user's motor skills, problem solving strategies, and cognitive organization.

The problem is, how do we evolve such a "fit" in any structured, or methodological way? Atwood et al (1979), Ramsey et al (1978), and Ramsey (1979) provide annotated bibliographies covering much of the relevant literature. In addition, Martin (1973) and Shneiderman (1980) both provide useful texts which address the problem. In spite of this work, however, it is important to recognize that the literature still falls short of providing the designer with guidelines sufficient to enable him to predict the effectiveness of' one interface design compared to another.2 The design of' the user interface is still an "art" rather than a science (Baecker, 1979)

Faced with a problem whose solution can not be derived from the literature, the designer of the human interface is confronted with two alternative strategies. On the one hand, a scientific approach can be followed, in which formal experiments are run, in an attempt to fill in existing gaps in our knowledge. On the other hand, an engineering approach can be taken, in which some ad hoc strategy is followed in order to deliver a system which was typically needed "yesterday". Due to pragmatics, the latter approach is by far the most prevalent although work such as Barnard et al. (1981) is an important example of the former.

Our objective in this paper is to present an approach to design which, on the one hand takes the pragmatic engineering approach, but at the same time attempts to accumulate data which will help form the basis for a more scientific understanding of the problem area. In so doing, we will draw examples from a case study in computer music, an appropriate study which has served as the test bed for many of the concepts presented.

2. ITERATIVE DESIGN

In attempting to design a system to "fit" the end user, behavioral issues must be considered and understood. Given the limitations of the analytical tools available, and our inability to adequately predict system performance in "real-world" situations, it is unlikely that the first implementation of any user interface is going to function as well as it could or should. Under these circumstances, an alternative is to take an iterative approach to design: keep trying until you get it right. However, two problems become immediately obvious. First, how do you know when you have got it "right", and why it is "right". Second, how can such an iterative approach be made practically and economically feasible?

In the approach described, we think of each iteration of a design as being a prototype whose purpose is to test a critical mass of the overall problem. On implementation, each prototype is tested by "guinea pig" users whose performance is monitored. Based on this experience, the performance of the prototype is evaluated, and the next iteration planned. Returning to the general questions posed above, we see that the successful use of the iterative approach is intimately linked to the following three issues:

- What is to be prototyped, and how?
- What is observed, and how?
- How are results evaluated, and subsequently applied?

It is impossible to give a simple solution to any of these questions; however, our experience over the past few years leads us to believe that there are certain general approaches which can be taken in each of these areas, whose cumulative effect is rendering the iterative approach viable in many applications. The next three sections will discuss these issues in more detail.

3. PROTOTYPING

3. 1 Introduction

The question of what to prototype relates to classic problem reduction. The designer must reduce the problem space into an ordered set of manageable sub-problems. Some of these will require testing in prototype form in order to be properly dealt with. To be as focussed and as efficient as possible it is essential that the designer have the ability to isolate what constitutes a "critical mass" of the problem under investigation. In addition, it is essential that the designer develop, and have at his disposal "prototyping tools".

Prototyping tools are software tools and modules which facilitate the design, implementation, and maintenance of application software. They are the tools, without which, an iterative approach to design would be impossible. From our point of view, the designer should strive towards developing his programming environment to the point where with any problem, each reasonable alternative is equally accessible for testing. The trap of the "path of least resistance" should be avoided, thereby reducing system imposed biases.

We feel that at any point that a design must be accepted because of the expense of testing a plausible alternative, attention probably should be redirected to the development of better prototyping tools, rather than prototypes themselves. The remainder of this section is devoted to a more detailed discussion of such tools, and how they function at different levels.

3.2 System Level Tools

3.2.1 Introduction: Too often the progress of design efforts has been severely impeded because of a lack of recognition of the importance of viewing the programming environment as an integrated whole. Systems are adopted because they have the "latest" technology in some area, with little or no thought as to what they support in the way of text editors, graphics packages (such as GPAC: Reeves, 1978), programming languages, or debugging tools (Crossey, 1977). But the suitability of any particular system can only be evaluated in terms of its ability to support experimental programming. This is an issue which is discussed more fully by Deutsch and Taft (1980). For a specific example, we will focus on one case: the influence of high-level languages on the effectiveness of the prototyping environment.

3.2.2 High Level Language: Most problems to be prototyped involve complex concepts. In order to be economically encoded, it is essential that the language used permits the succinct expression of the concepts to be tested, and their inter-relationships. Many of the problems of interest to the Human Factors researcher present real problems when it comes to selecting a suitable high-level language for their investigation. Take, for example, the problem of specifying the user interface and causal relationships for some complex real-time control task. As driving an automobile demonstrates, the human operator is capable of coordinating the concurrent expression of several channels of information. However, if the receiving end of these parallel data channels is a digital computer, we are hard pressed to find a high-level programming language which will facilitate the terse and concise specification of how these data are to control some ongoing process.

There are various consequences to the above observations. First, inappropriate specification/implementation languages constitute a major stumbling block in computer-based Human Factors research. Languages make a significant difference, and should be carefully considered by the researcher. As Iverson (1980) points out, notation is an important tool of thought. Second, research into languages which support communicating parallel processes, data-driven processes, graphics interaction, real-time event scheduling, and the automatic monitoring of user actions, is of critical importance. Baecker (1979) and Green (1980) are examples of our ongoing work in this domain.

3.3 Application Level Support

3.3.1 Introduction: The basis of a good prototyping environment is the development of a set of well defined modules, or "building blocks", which facilitate the efficient construction and testing of prototypes. The adoption of such a tool-building paradigm is facilitated in recognizing that in most "real-world" situations prototyping is restricted to a limited number of classes of problem. We can often isolate recurring types of transactions for which high-level support (or "breadboarding”) modules can be provided to the applications programmer, thereby facilitating his ability to implement a particular system. For purpose of illustration, we present examples of such modules developed in the course of our research into computer music Systems.

3.3.2 Menu System: Menu-based interaction is an important, recurring component in human-computer dialogue. It is accompanied by two main classes of problems. First, there is the graphics design problem. This encompasses menu layout and design. Second, there is the problem of defining the cause-and-effect relationships that occur as the result of interaction with the menu in various contexts. Unless supported with adequate tools, the undertaking of these two tasks not only retards the prototyping process, but the resulting low-level complexity consumes design resources which are limited, and which should be directed towards application-oriented problems.

Recognizing this problem, we have invested considerable energy in designing, implementing, and documenting a system which can serve as a template for menu-based interaction (Buxton, Reeves, Patel, and 0'Dell, 1979). In using this system to implement menu-driven software, the designer need only "plug-in" pertinent values, and the system will automatically handle initialization, event detection, and command invocation.

The menu system is an interesting example of how a prototyping environment can be used to "bootstrap" itself. That is, the package has been expanded and refined as observation and evaluation have provided insights into the nature of menu based dialogue.

In terms of the two problems of menu based interaction (layout and definition of causal relationships), the current menu system addresses the latter. To be complete, we must now pursue the issue of developing a tool which allows the user to apply techniques of computer aided design in menu layout, thereby generating the values to be "plugged into" the template provided by the existing module. Thus, through an iterative approach, we see how our reach in prototyping concepts efficiently has been extended: a high-level language was used to implement the graphics package, which was used to implement the existing menu system, which will be used to implement the menu layout system, which will be used to efficiently implement application-level software. Each step serves to improve the applications programmer's ability to undertake his task. In all of the above, the key point that emerges is the importance of balancing efforts between the development of prototypes, and the development of prototyping tools, and using the same approach of iterative design in each.

3.3.3 Directory Windows: One recurring problem which we have observed with computer naive users has to do with retrieving previously defined data, or files. This problem is manifest in different ways. First, having learned to associate particular data with a unique file name, the user has difficulty as a result of the ambiguity of the possibility of two co-existing versions of that data, one in primary memory, the other on disk. On retrieval, for example, which data is referenced? This conceptual problem is further complicated by difficulties and awkwardness arising from having to remember file names, remembering their spelling, isolating files of a particular type from among the many types which may exist, typing file names, and difficulties in "browsing" through files.

All of these problems share a common trait: they place a burden on the user's memory and cognitive structures, a burden which is completely secondary to the task for which the computer was originally adopted. They are problems of means, rather than content, and consume vital resources which should be allocated to those problems which are the user's real concern.

These problems cut across many application areas, and for many classes of systems, common modular tools can be applied to their elimination. The resultant benefit to the user is a more convivial interface, and to the designer, a set of high level building-blocks which facilitate the implementation of prototype systems.

Our approach to developing such modules is based on what we have called "directory windows". A directory window is a region on the user's screen which allows files of a specified type t6 be viewed and accessed. The code which supports these windows consists of a set of well documented modules which allows the programmer to specify the size and placement of the window on the screen, the type of file onto which the window looks3, the (alternative) means for scrolling through and selecting from the files, and the source and representation of the files in the window. The examples which follow illustrate these points, and relate them to the problems associated with file retrieval which were outlined at the start of this section.

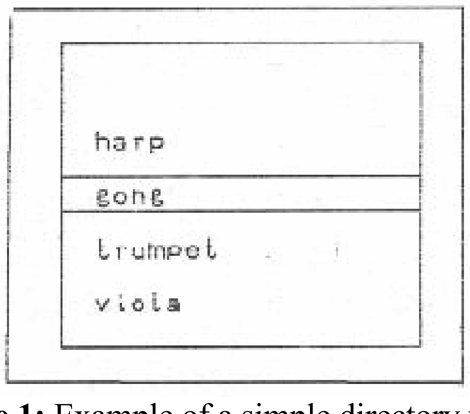


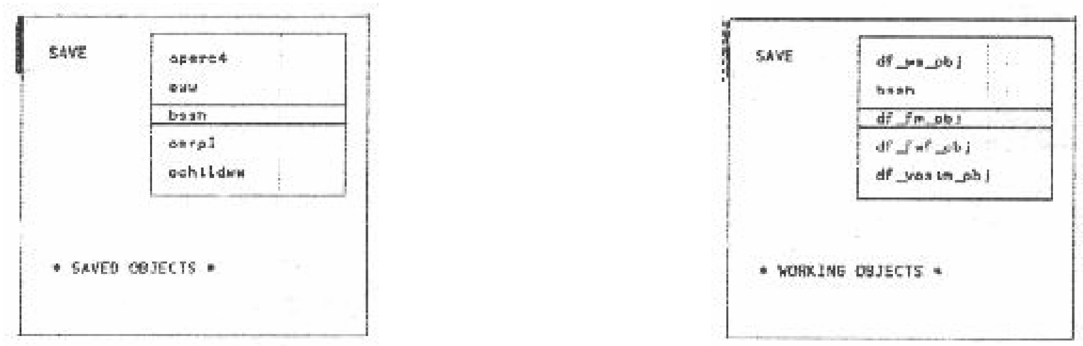
Figure 1: Example of a simple directory window

In its simplest incarnation the window is a rectangle containing a list of file names, one of which appears between a central pair of horizontal lines. Each name is that of an existing file. Significantly, all files whose name appear in a particular window are of a specific type: for example, musical scores. The file whose name appears between the horizontal lines is known as the "current" file. In the context of a score editor, therefore, the name between the horizontal lines would be the name of the score being edited.

Continuing with the example, different files can be loaded into the editor by causing their names to appear between the horizontal lines. This can be accomplished by various means, including pointing at a name visible in the window4, or using some transducer to "scroll" names through the window, thus accessing file names previously outside the window.

The importance of this mechanism, besides its generality, lies in the fact that the user can access files without having to remember names or spelling, and without cumbersome typing. As well, the applications programmer is provided with a mechanism to restrict data presented to only that which is relevant in a particular context, since all files are keyed by type.

A problem already alluded to derives from the ambiguity which results from allowing two different versions of a file to co-exist - one in primary memory, and one on disk. Through the use of the "switch" illustrated in Figure 2, a self consistent method is provided to help deal with the problem.



(a) Accessing scores from disk. (b) Accessing scores from primary memory.
Figure 2: Directory Window Switches.

Namely, with the switch position reading "Saved Scores" , as illustrated in Figure 2a , we have an explicit message indicating that the files onto which the window looks are disk files which have been previously saved. On the other hand when the switch reads "Working Scores", as illustrated in Figure 2b, the user is provided with an explicit reminder that the files onto which the window looks are temporary working versions, stored in primary memory. This user-controlled switch provides a means of retrieving either the working or saved version of a file, as well as a mechanism for verifying whether a particular file has been saved.5

A final point relates to how directory windows facilitate browsing through files, and how certain types of files can be selected according to contents, rather than (an often arbitrary) name. This is seen in Figure 3, where an iconic representation of the file data (in this case time-varying functions) is provided in addition to the file name.

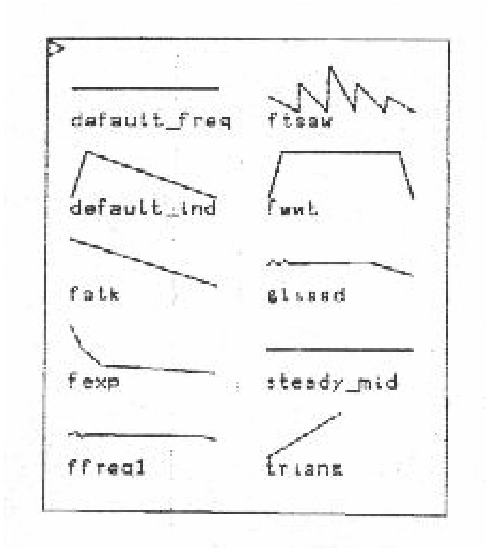


Figure 3: Iconic Representation of File Contents

The last example illustrates how the concept of a directory window is independent of external representation, in spite of the fact that a self-consistent method of interaction is maintained. This is an observation which can be, and should be, developed in future research. Our experience has shown that an appropriate iconic representation of file data substantially improves the user interface in many real-world contexts. Prototyping tools should, therefore, expedite experimentation in this regard.

3.3.4 Graphic Potentiometers: Generically, music composition and performance fall within the realm of computer aided design and control applications. One type of transaction observed to be recurring in these applications is the adjustment or setting of scalar values, or parameters. That is, the kind of transaction that might involve turning a knob, adjusting a potentiometer, or typing a number. Since this is a recurring transaction, we have developed another prototyping tool which, again, facilitates "bread-boarding" systems by providing well documented, highly parameterized building-block modules. These modules, we call graphic potentiometers.

Three graphic potentiometers are shown in Figure 4.

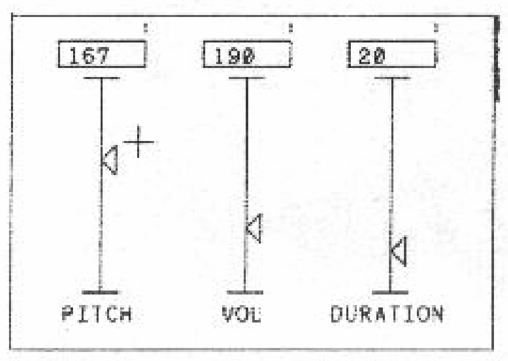


Figure 4: Simple Graphic Potentiometers

Visually, they represent a physical potentiometer whose current value is shown in an analogue (high-low) manner, by the position of a triangle representing the "handle". The setting can be adjusted by dragging the handle up or down using some pointing device, or alternatively, any other transducer supported by the graphics package. On adjustment, two things happen. The visual display is updated, as is the parameter associated with the potentiometer - both in real-time. Thus, using an example from the figure, if there was a tone sounding while the pitch potentiometer was being adjusted, the resulting acoustical change would be heard.

Thus far, the graphic potentiometer has been discussed in only analogue terms. Notice, however, that a numerical read-out of the current value of the parameter appears in the box above a potentiometer. As the parameter is adjusted, this read-out is also dynamically updated. Furthermore, indicating the box with the pointing device enables the user to directly specify a value by typing. This is illustrated in Figure 5. There are two important points to be made in this example. First, the "tracking symbol" which was a "+" in the previous example has now become an icon representing a terminal. This provides a visual prompt that a value is to be typed. Second, while the value is being input, the digits being typed appear directly within the indicated box (rather than at the bottom of the screen to be posted in their final position on completion). Both features have the important property that they provide visual feedback and messages at that location on the screen on which the user's attention is currently focussed.

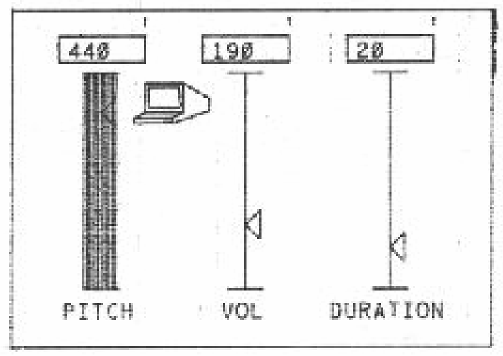


Figure 5:

Our observation is that this property, which is too often absent, is extremely effective in reducing operator errors.

3.4 Summary

The concept of prototyping tools has been introduced. Examples have been given from both a general systems, and an applications specific level. The point emphasized was that concentrating on developing a suitable prototyping environment will facilitate pursuing an iterative approach to design. Furthermore, through the provision of good prototyping modules, the designer is better able to concentrate on the principal problems under investigation, rather than becoming trapped in a morass of low level secondary issues. Finally, it was observed that the iterative approach of prototype-observe-evaluate can be applied to prototyping tools themselves, thereby allowing the prototyping environment to be bootstrapped in a cost-effective way.

4. OBSERVATION

4.1 Introduction

The second key aspect of the iterative approach concerns techniques for collecting data for interface evaluation. An implicit assumption in our discussion is that the interfaces to be evaluated have been implemented in prototype form, and are to be tested under pseudo "real-world" conditions. Again, the point is that until improved predictive and analytical models are developed, we can only determine the relative effectiveness of alternative interface designs through the observation of test subjects performing representative tasks.

The appropriateness of alternative techniques for observation are obviously highly dependent upon what is being tested, and how it is to be evaluated. In order to better understand the key alternatives available, we devote the remainder of this section to their enumeration and characterization. The discussion will, by necessity, be cursory in nature.

The techniques discussed divide into two main categories -- those which do not involve verbalizations by the subject, and those which do. The first group is used to directly observe the behaviour of test subjects; however, the great deal of mental activity and knowledge which the subject employs when using the prototype cannot be observed directly. This is where the group of verbal techniques plays a role (Bainbridge, 1979). In order to try and learn how and why a user makes decisions, it is necessary to minimize the distortions introduced in the user's behaviour by having to give a verbal report. Similarly, it is necessary to minimize the distortions in the report itself as a description of the user's behaviour.

4.2 Non-Verbal Techniques

4.2.1 Observation by Researcher involves a researcher standing over the subject and simply watching the interaction. It allows the human observer to catch basic low-level problems in the interaction which the user might not be aware of due to concentration on the higher cognitive levels of the task being performed.

The way in which software expects the digit ‘1' to be specified can serve to illustrate one type of insight to be gained through this approach. A researcher watching a user's operation of the teletypewriter might notice a predisposition on the subject a part to enter, instead, the letter 'l', due to previous experience with a common typewriter. Even if this happened repeatedly, it might not come to the researcher's attention except by direct observation, since the subject might not ever report it.

A drawback of using this technique regularly is the practical problem of having researchers available constantly at sessions.

4.2.2 Observation by Video Tape provides a more permanent record of the session, one which may be viewed repeatedly at the researcher's leisure. This technique is obviously more expensive. As well, it poses the problem of having to decide which aspects of the interaction to film. If the system involves input and output transducers spread over a relatively large area, then all aspects of the interaction could not be easily filmed simultaneously.

The technique has been used by Card et al (1980), and at MIT by Bamberger[6]. In the latter example, the technique was used to observe children carrying out a musical problem solving task involving the arrangement of a set of pitched bells. The children are videotaped while they perform the task and at the same time, a researcher asks the child what he is doing and the reasons behind the action. After the session, the videotape and tape-recorded responses are analyzed to try and determine the child's cognitive organization which guided his behaviour during the task.

4.2.3 Observation by the System is particularly efficient since it involves collection of session data by the system itself. The most basic example of this is a non-graphical dialogue on a teletypewriter, which results in an exact transcript of the session. More generally, modules embedded in the system can keep records of the interaction (perhaps time stamped) for future analysis by the researcher. These modules themselves can be considered to be prototyping tools, as they are used to facilitate the development of application software by providing a convenient method of data collection. A drawback of this technique is that it does not afford a way of evaluating the kinesthetics of the interaction.

An example of this technique is an extension of the menu system outlined in Section 3.3.2. As each item of the menu is chosen, the system can log the time, the name of the menu item, and its screen location. The tool provides a record of the order and timing of menu item activation, as well as a graphical display of the user's hand motion.

This is clearly a graphics tool which can be used in a number of ways. It can give a clear view of areas of the menu which are used to a great extent, thereby providing insights into a more effective positioning of menu items. It can be used to help develop other tools which would enable a user's session to be animated" or "played back" for subsequent analysis. However, there is still a great deal of work which must be done before the data provided by this tool can be exploited to its fullest. For example, it is not clear that if there is heavy traffic between two menu items, that those two items should always be in close proximity.

4.3 Verbal Techniques

4.3.1 Report of Impressions by the Subject involves the subject reflecting on specific aspects of the session. The impressions can be expressed orally (to an attendant researcher or tape recorder) , or they can be written. They can be expressed either during or after the session.

Introspection during a session is advantageous for obtaining fresh accurate detail from the subject. If spoken and recorded ,it can be neatly synchronized with aforementioned techniques of observation. If written, the comments can be assisted by any system facility which allows messages to be recorded. In this case messages can be time stamped in order to place them in context during analysis. The danger with any explicit reporting by the user during the session is that it distorts timing data for task performance, and diverts the users attention from the applications task at hand.

Recording impressions after a session allows time for user introspection leading to a more coherent view of the session as a whole, rather than as a series of individual interactions.

4.3.2 Detailed Session Commentary by the Subject involves the subject keeping an explicit record of his actions and thought processes during a session. Either written or verbal, the subject maintains a running commentary on what he would like to do, how he is going about doing' it and why. This type of record allows a researcher to discover points at which the interaction is unsatisfactory, even if the subject is unaware that there is a problem. These types of problems are not readily discovered by the previous technique, which only records user perceived problems and points of interest. We distinguish between the two approaches, report of impressions and detailed commentary, since each provides a different kind of data. The former highlight specific problem points, the latter addresses more global aspects of the user's problem-solving strategies.

A drawback of detailed commentary is that the amount of detail needed puts a heavy demand on the user during the session, a point which must be taken into account when designing tasks to be observed using this approach. One alternative is to have the subject provide the commentary while viewing a play-back of the session on video tape. Finally, as Bambridge (1979) points out, one must be careful in making assumptions about the user's knowledge, based on such commentaries.

4.3.3 Interrogation by the Researcher provides a means of minimizing distortions arising from false conclusions derived in analyzing user reports.

Questions need not be posed during the session. They can be asked during or after, either verbally or in writing. Well placed questions can help to quickly clear up confusion on the part of both the subject and researcher.

Questionnaires are useful tools in interrogation, and have been used by Barnard et al (1981), and Dzida et al (1978) in studying the user interface. The advantage in this approach is that it provides results which are more structured, and easier to analyze than the free-form commentaries of the other verbal techniques discussed. They are, however, less adaptable to circumstances.

4.4 Summary

A set of techniques have been presented which can be used to collect data concerning the behavior of a subject carrying out specific tasks using an interactive computer system. Combined with working prototypes of various user interface designs, the basis for an environment for interface evaluation and comparison is provided. The prime question remaining concerns establishing criteria for evaluation, in order to guide the determination of what data is to be collected, and how.

5. EVALUATION

Our objective is to develop the ability to design effective user interfaces in a consistent and methodological way. As a prerequisite, we must first refine our abilities to quantitatively compare and evaluate designs. This we view in two contexts: first, through the evaluation of data collected during benchmark tests involving subjects performing tasks on prototype systems; second, evaluating designs before implementation, according to models derived from experiment. We see the iterative approach as a means of carrying out the former, so as to achieve the latter. As the experimental approach derives improved models, the need for iteration will become eliminated, or be able to be applied to higher-level problems of the user interface.

Card et al (1978) is a good example of the approach. The study investigated alternative techniques for pointing at, or selecting, text items displayed on a CRT. Building on these results, another study was carried out (Card et al 1980) which resulted in a model known as the "keystroke model" , which (within a restricted context) has proven effective in predicting user performance. The argument could still be made, however, that techniques other than testing under real-world conditions could b~ used to derive the results to refine such models. While this is undoubtedly true in some cases, there are some real pitfalls of which one must be aware. First, one of the prime benchmarks in determining the effectiveness of a user interface concerns temporal response. It is difficult to imagine many alternative techniques which take adequate account of this parameter. Second, studies such as Barnard et al (1981) have demonstrated that (even expert) users are often unreliable in their prediction of the "goodness" of simple interfaces when presented with the design on paper, prior to implementation. Even if tests are run on actual systems, the subjects' perception of the relative goodness of a design is unreliable when compared to alternative measures of performance. This is demonstrated in Moses and Maisano (197~), for example. The purpose of the study was to examine user performance in finding the shortest route between two cities, when presented with different methods of map display. The alternatives differed in how the change from one displayed map to another was effected: either discrete changes with 0, 25, or 50 percent overlap, or continuous changes The result of interest here is that while most subjects preferred the continuous alternative, that alternative consistently resulted in the longest solution time to the problem!

Based on the above, it is clear that often we must not only test designs under simulated real-world conditions, we must also develop well defined criteria for measuring performance. Dzida et al (1978) conducted a study to establish such criteria. The results are summarized below:

- Self-Descriptiveness:
  - transparency of dialogue organization and dialogue sequence at any time;
  - clearly arranged presentation of system functions;
- User Control:
  - process canceling possible without detrimental side effects;
  - command language syntactically homogenous;
- Ease of Learning:
  - user manuals superfluous;
  - no special data processing knowledge needed to use system;
- Problem Adequate Usability:
  - free formatted command input accepted;
  - minimal need for the user to perform clerical or housekeeping activities;
- Correspondence with User Expectations:
  - similar system behaviour in similar situations;
  - feedback provided to enable user to recognize effects of his input;
- Flexibility in Task Handling:
  - system messages with different levels of detail dependent on user status;
  - shorter ways for trained users to perform tasks;
- Fault Tolerance:
  - only partial retyping required if previous input was erroneous;

    - typical typing errors tolerated.

To these, we would add two (rather surprising) omissions:

- Time Factors:
    - response time for different types of functions;
    - time required to undertake a given task;

- The Number of Errors Typical in Performing Representative Tasks

Even if we accept the above list as complete, and develop the ability to assign a quantitative measure for each when examining a particular design, we still fall short of being able to provide practical tools to the designer of the user interface.

First, in any real environment there is a complex relationship among these different measures. For design purposes, we either have to restrict our predictive model to one or two measures (such as task performance time in Card *et al*., 1980), or develop our ability to analyze the transactions encompassed within a particular interface according to the cross-relationships and relative weights of these criteria. The importance of the latter is seen in a study undertaken in our lab (Hogg and Sniderman, 1979). The objective was to evaluate and characterize alternative graphical techniques for specifying the pitch/time information of notes making up a musical score. When measured against the criteria outlined above, each technique tested came out with different results. The point to make, however, is that out of context no "best" solution can be identified. Each has its own characteristics: one is well suited for CAI, another for composition. Given an application, how then can we analyze its constituent transactions in terms of their requirements with respect to there criteria, and how can we use these results as a guide in choosing the most effective way of implementing the user interface? The answer is probably yes, but only using an *ad hoc* approach. Furthermore, on moving into application areas which are less familiar, the absence of a well structured model for interface design would precipitate far less satisfactory results.

The above example raises the second problem: one based on pragmatics. Even if we can derive a quantitative measure of how well a particular interface works, we are still restricted in our ability to determine why it works, and to be able to use that knowledge in designing future systems. Again the problem is one of inadequate models, and one which will in an incremental way, be addressed by the iterative approach.

6. CONCLUSIONS

Rather than give solutions, or results, the thrust of this paper has been to point out problems. In particular, we have focused on lack of scientific understanding of the human-computer interface in interactive systems. This absence has been seen in the lack of effective models which can be used by the designer in order to predict and evaluate the performance of interface designs. As a means of evolving such models, an iterative process has been proposed. Central to the approach was the active participation of test subjects in experiments. To support such testing, and render the approach economically and practically viable, three things were deemed necessary: suitable prototyping tools, techniques of observation, and methods of evaluation. Based on our own work, and that of others reported in the paper, we believe that the approach provides a valuable tool for extending our understanding of the user interface, if for no other reason than it helps to focus our attention on those areas in which there are critical gaps in our knowledge.

7. REFERENCES AND BIBLIOGRAPHY

Atwood, M., Ramsey, R., Hooper, J. & Kullas, D. (1979). *Annotated Bibliograohv on Human Factors in Software Development*. Alexandria, VA., U.S. Army Research Institute.

Baecker, R.M. (1979). Human-Computer Interactive Systems: A State-of-the-Art Review. Presented at the second International Conference on Processing of Visible Language, Niagara-on-the-Lake, Canada.

Baecker, R.M. (1980). Towards an Effective Characterization of graphical Interaction, in Guedj, R.A., ten Hagen, P., Hopgood, F.R., Tucker, H. and Duce, D.A. (Eds.), Methodology of Interaction, Amsterdam: North Holland Publishing, 127-148. .

Baecker, H., Buxton, W. & Reeves, W. (1979). Towards Facilitating Graphical Interaction Some Examples from Computer-Aided Musical Composition. *Proceedings of the 6th Man Computer Communications Conference*, Ottawa,
Canada.

Bambridge, L. (1979). Verbal Reports as Evidence of the Process Operator'sKnowledge. IiL~. ~. 1i~XL-Machine Studies 11: 411-436.

Barnard, P., Hammond, N., Morton, J., Long, J. & Clark, I. (1981). Consistency and compatibility in human-computer dialogue. *International Journal of Man-Machine Studies* 15(1): 87 - 134.

Buxton, W., Reeves, W., Patel, S., & O'Dell, T. (1979). *SSSP Programmer's Manual.* Unpublished manuscript, SSSP/CSRG, University of Toronto.

Card, S.K., English, W.K., & Burr, B.J. (1978). Evaluation of Mouse, Rate-Controlled Isometric Joystick, Step Keys, and Text Keys for Text Selection on a CRT. *Ergonomics* 21(8): 601-613.

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(7): 396-410.

Crossey, S. (1977). *An Interactive Graphical Source Language Debugging Svstem*. M.Sc. Thesis, University of Toronto.

Deutsh, L. P. & Taft, E. A. (1980). Requirements for an Experimental Programming Environment. *Xerox PARC Technical Report CSL-80-10.*

Dzida, W., Herds, S. & Itzfeldt, W. D. (1978). User-Perceived Quality of Interactive Systems. *IEEE Transactions on Software Engineering.* SE-4(4): 270-276.

Encarnacao, J. & Tozzi, C. (1979). *Seilliac II Bibliographv Report 1*. Darmstadt: Technische Hochschule, Fachbereich Informatik.

Green, M. (1980). *The EDL Programming Language.* Unpublished manuscript, DGP/CSRG, University of Toronto.

Guedj, R. & Tuckse, H. (Eds.). Methodologv in Computer Graphics. Amsterdam: North-Holland Publishing Company.

Guedj, R.A., ten Hagen, P., Hopgood, F.R., Tucker, H. and Duce, D.A. (Eds.), (1980), Methodology of Interaction, Amsterdam: North Holland Publishing.

Hogg, J. & Sniderman, R. (1979). *Input Tools Project Report.* Unpublished manuscript, SSSP/CSRG, University of Toronto.

Iverson, K. (1980). Notation as a Tool of Thought. *Communications of The ACM*. 23(8): 444-465.

Martin, J. (1973). *Design of Man-Computer Dialogues*. Englewood Cliffs: Prentice-Hall.

Moses, F. L. & Maisano, R. E. (1978). User Performance Under Several Automated Approaches to Changing Displayed Maps. *Proceedings of ACM SIGRAPH*, 12(3): 228-233.

Ramsey, H. R., Atwood, M., & Kirshbaum, P. (1978). *A Criticaly Annotated Bibliography of the Literature of Human Factors in Computer Systems*. National Technical Information Service.

Ramsey, H., R. (1979). Human Factors in Computer Systems: Review of the Literature and Literature and Development of Design Aids. *Technical Report SAI-79-113-DEN*, National Technical Information Service.

Reeves, W.T. (1978). A Device-Independent, General-Purpose, Graphics System in a Minicomputer Tine-Sharing Environment. *Technical Report CSRG-93*, University of Toronto.

Sheridan, T. & Ferrell, W. (1974). *Man-Machine Svstems: Information, Control, and Decision Models of Human Performance.* Cambridge: MIT Press.

Shneiderman, B. (1980). *Software Psycholgy: Human Factors in Computer and Information Svstems*. Cambridge: Winthrop Publishers, Inc.

Ting, T.C. & Badre, A.N. (1976). A Dynamic Model of Man-Machine Interactions: Design and Application with an Audiographic Learning Facilty. *Int. J. Man-Machine Studies* 8: 75-88.

Treu, S. (Ed.) (1976). *User-Oriented Design of Interactive Graphics Systems*. New York: ACM.

Footnotes

[1] The work reported in this paper has been undertaken as part of the Structured Sound Synthesis Project of

the University of Toronto. This research has been funded by the Social Sciences and Humanities Research Council of Canada, whose support is gratefully acknowledged.

[2] There are a few notable exceptions, however, such as Card, Moran, and Newell (1980), which provide predictive models which work well in restricted contexts.

[3] In this there is the implicit understanding that all files are of a specific type. This is not at all unreasonable or difficult in most contexts, and can result in substantial benefits to both the user and programmer, the motivation being much the same as for strong typing in programming languages.

[4] The cursor of a digitizing tablet, or a light pen are two possible pointing devices.

[5] One beneficial attribute of this mechanism is that in the context of an editor, it facilitates the ability to have more than one "working" scores in primary memory at a time. This is in contrast to most text editors, for example, which restrict the user to working on only one file at a time.
[6] Personal communication.