Human–Computer Interaction – INTERACT '87
H.-J. Bullinger and B. Shackel (Editors)
Elsevier Science Publishers B.V. (North-Holland)

EVALUATING USER INTERFACE COMPLEXITY

John Karat, Richard Fowler and Mary Gravelle

IBM Austin

This paper presents an attempt to utilize a formal model in the study of user interface development. A study was conducted to examine learning and performance differences between a command language and a direct manipulation system. Subjects initially unfamiliar with computer systems learned file management functions and carried out a series of tasks on one of the systems. Experimental results point out large differences in performance between the command language and direct manipulation systems which favor direct manipulation. Formal models of the knowledge required to use the systems were developed following the framework suggested by Kieras and Polson [1]. There are difficulties in mapping predictions from the formal models to the experimental data for the systems involved. Analysis suggests that inability of the formal model to predict error data was a basic problem with the formal analysis.

1. INTRODUCTION

Considerable effort has been directed toward development of formal user interface analysis tools. Though perhaps for different reasons, attempts to 'harden' the science of human computer interface design have been called for from both the development (seeking to increase quality and lower development cost) and scientific (using computer systems to study human cognition) communities. There is some reason to believe that some important steps are now being taken in this area (Newell & Card [2]). From the GOMS model of Card, Moran and Newell [3], to extensions of this work put forward by Kieras and Polson [1], we are seeing a number of techniques aimed at improving the design process for user interface development. These techniques claim to provide assistance that will enable accurate design decisions to be made, without necessitating costly user testing. They do not necessarily claim to eliminate the need for iterative testing of the design to fine tune the details, but they do hold a promise of an additional tool for the interface designer.

Faced with the task of designing the user interface for a new system, there are many design decisions that one is faced with. Clearly the most important step in the process is to carry out a careful analysis of the task which the system will be used to perform (the often cited, but not so well understood 'task analysis' phase). Once this has been accomplished much still remains to be decided. Computer systems can vary in a great number of dimensions, and the designer is faced with a number of decisions. While it might be desirable to provide a set of standard answers for user interface design questions, the closest we come now is in the form of general guidelines (many such guidelines exist under the name of standards, architectures, or guidelines). Decisions about many aspects of interface design will continue to vary depending on the task and users involved. Placing useful tools in the hands of developers will likely play a larger role in improving the quality of user interfaces than the development of standards.

There has been some promising preliminary work which suggests that a formal analysis of knowledge required to learn and use a system would be useful in making design decisions. Kieras and Polson [1] have shown that a variety of user behavior can be accounted for through production system modeling. The framework described by Kieras and Polson [1] is an extension of the GOMS work developed by Card, Moran, and Newell [3], that is intended to provide predictions for ease of learning and transfer of knowledge in addition to the predictions of ease of use of a computer system available from the keystroke model. In the Kieras and Polson

work, knowledge necessary to use a system is modeled in a production system. Predictions of ease of learning are obtained through a count of the productions contained in the model, and predictions of transfer between systems (or components of systems) are obtained from the number of shared productions. Performance predictions are obtained from counts of production cycles when carrying out tasks. Considerable experimental work has been conducted in attempting to validate the predictions of this framework (Polson and Kieras [4], Polson, Muncher, and Engelbeck [5], and Ziegler, Hoppe and Fahnrich [6]).

The development of production systems, like the development of any cognitive model, is not a completely well defined task. At this point, it certainly is not something that would be easy for an experienced software developer to carry out. In an applied setting, issues critical to this framework (such as maintaining a consistent production granularity) might even be considered problematic for experienced cognitive psychologists. Experience in model building often leaves one wondering if certain aspects of the task are receiving too much or too little attention, and one is always aware of the fact that and model developed is not unique. While this is a lesser problem in many areas of modeling (it is of little trouble when one is only seeking a sufficient model of a single system or cognitive task is being modeled), it is a problem if two quite different systems are to be compared and if the heart of the predictions is based on a count of productions involved. Another issue of immediate concern here is whether or not such analysis can capture difficulties (primarily error recovery) which occur in more natural settings than the laboratory work of Kieras and Polson. There are many other issues in cognitive modeling through production systems, but these are the central issues for this investigation.

1.1. The Design Question

There has been movement to change the basic style of interaction with computer systems from interfaces dominated by command languages in which the user types fixed format command strings to so called "direct manipulation" interfaces in which the user manipulates displayed objects to carry out some function. While claims for the superiority of the direct manipulation style of interface abound, there has been little formal demonstration (through experimental or theoretical work) of the benefits of such systems. The situation is complicated by the fact that a mixed bag of user interface components (e.g., use of high resolution graphics, various pointing devices, and employment of system metaphors) have been lumped together in discussing direct manipulation or graphical user interfaces. The experiment here compares an interface low in the components associated with the new user interface style with one which includes several features which seem important to these interfaces.

As a starting point for our examination, we consider simple file management in a computer system with a hierarchical structure. The structure considered is one in which the system has a root directory which may contain files and other directories (any directory may contain both files and other directories). The contents of a directory may be listed (the names of the entries in the directory may be displayed), the contents of a file may be listed, and files may be copied or deleted. The necessary actions in the system are identifying the action to be carried out (list directory, list file, copy file, and delete file), and identifying the object of the action (a single file or directory, except in the case of copy which requires a file and a destination directory).

Given that command language and direct manipulation systems are clearly different, a key question is whether or not there are formal analysis techniques which would help the designer decide between such alternatives. One question examined here is whether or not a production system analysis of the two systems would predict one as superior to the other. To examine this we carried out both a "typical" experimental comparison of users learning and using these two systems, and a formal analysis of the complexity of the two systems.

2. SYSTEMS TESTED

For this work, we provided subjects with an existing file structure, and asked them to carry out a restricted set of manipulations on this structure. The two systems compared include a standard command language system (PC/DOS) and a prototype direct manipulation system developed to run on the same hardware (with the exception of the inclusion of a mouse). The systems both used an IBM Personal Computer

with monochrome display. DOS includes commands for file manipulation which are highly descriptive (COPY, TYPE, DELETE) and some which are less so (DIR and CD). File and directory names have a restricted syntax, and files are contained within directories. Directories may contain other directories resulting in a hierarchical structure. In DOS, a command is entered by specifying the action name, and the name of the object (in some cases the object may be implied, or in the case of COPY, two specifications for object may be needed). The prototype system (called simply GPH), presented the directory structure in a tree fashion (using only character graphics - enabling horizontal and vertical lines to create a directory "tree"). Pointing to a directory name and clicking a mouse button twice will 'open' the selected directory, and list its contents (file names) at the side of the display. Pointing and double-clicking a file name will provide a listing of the file contents (text). Pointing to a file or directory, pressing and holding down a mouse button, and moving the pointer to a new location, caused the selected object to be copied or moved to a new location. In GPH, commands are entered by selecting an object by pointing to it, and then carrying out some action (clicking or dragging). The techniques are similar to those used in a number of commercial systems.

A hierarchical file structure with a total of 207 directories was used. For the GPH system the structure was presented in a viewport in the rightmost 64 columns of the screen taking up 25 lines of the display. In order to move from the top of the structure (root directory) to the bottom, subjects had to scroll a maximum of 125 lines by attempting to move the mouse pointer beyond the top or bottom of the display. This extensive structure was used to provide a non-trivial structure for the GPH system, and also to examine issues in file structure not discussed in this paper.

3. METHOD

Subjects in the experiment read introductory material describing basic file management functions, carried out practice tasks with each function, and carried out a series of tasks after they had completed practice with all functions involved.

3.1. Subjects

Twenty subjects were recruited from temporary employment agencies to be tested. All subjects were initially unfamiliar with file management tasks on personal computers (previous computer experience was accepted provided subjects did not have knowledge of operating system function). Subjects were randomly assigned to one of two experimental groups. The systems used were IBM DOS and a prototype direct manipulation system (GPH) developed for this test.

3.2. Equipment

In the experiment all subjects carried out a set of tasks on a standard IBM PC/XT. Instructional material provided to subjects was developed for this test. Standard reference material was not provided subjects in the experiment.

3.3. Procedure

Subjects were brought to the experimental room and told that they would be asked to carry out some tasks on the personal computer and that we were evaluating the system and not them. The experimenter provided the subject with a folder describing the tasks and left the room. Subjects proceeded at their own pace, and could call the experimenter if they encountered any problems. The experimenter would intervene when called for assistance by the subject, or when the subject had spent at least five minutes on a problem and was making no apparent progress in resolving it. An attempt was made to keep interventions to a minimum, while not overinflating the magnitude of specific problems. Subjects first read description of the general features of the system (keyboard, display, etc.). Following this subjects read a description of a hierarchical file structure which was designed to represent part of a local school system. The first level of directories were the names of two high schools. In each of these directories were files and directories for students, administration, teachers, courses, and sports. Additional breakdowns occurred in each of these directories with a maximum directory depth of seven levels. Subjects took as much time as they needed reading about the file structure in order to understand it. The description of the complete structure was available to them during the remainder of the experiment.

Subjects spent an average of about 30 minutes reading about the file structure.

When subjects felt comfortable with the file structure they continued reading about file management functions. The order of presentation of the functions was list directory, change directory, copy file, delete file and list file. For each function the meaning of the function were explained and several examples were provided which subjects carried out. Following the examples, subjects were asked to carry out several practice tasks with the current function. The tasks were designed to explore performance at different levels in the hierarchy. On completion of the practice tasks subjects continued on to the next function. When all functions were complete, subjects were asked to carry out 32 tasks which included a mix of function and depth in the file structure. On completing these tasks, the subjects were debriefed and allowed to leave. The DOS and direct manipulation task sets were identical with one exception, and subjects carried out the same practice and final tasks in the same order for each system.

4. RESULTS

Subjects in the experiment took between two and four hours to complete the tasks. In the DOS condition only five of the ten subjects completed the entire task set (means are reported for subjects completing each phase of the experiment), while all ten subjects completed the GPH condition. Data for reading instruction times for each function for the two systems are presented in Figure 1. As would be expected, there was no significant difference between the two conditions for time to read the directory structure information (STR in Figure 1). For the individual function instructions (list directory, copy file, delete file and type file contents) there was a significant difference for the directory listing and change directory function, and no difference for the remaining functions.

Data for performance on practice tasks (list directory, change directory, copy file, delete file and type file) and final tasks (change directory was not used) for the two systems are presented in Figures 2 and 3. The full bar height represents total task time (i.e., the time from first reading the task until it was correctly completed), and the shaded area of each bar represents correct taks (total time with error time removed). Correct time (which excludes only undetected subject errors such as incorrect file or directory specifications or mouse pointing mistakes) was measured from the onset of a correct trial until final entry. For all task sets there was a significant difference in total task time between the DOS and GPH groups favoring the direct manipulation system. Time spent in error trials is also significantly greater for the DOS subjects compared to the GPH subjects.

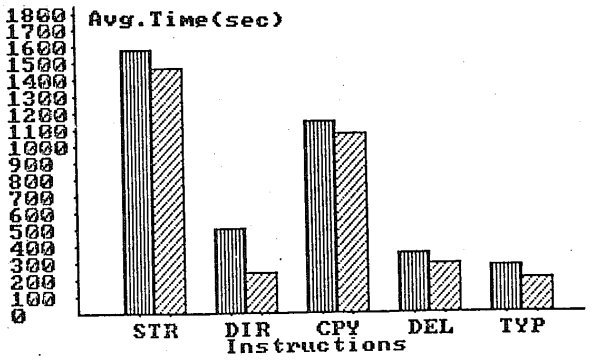

FIGURE 1. Instruction Reading Times by Type of Instructions

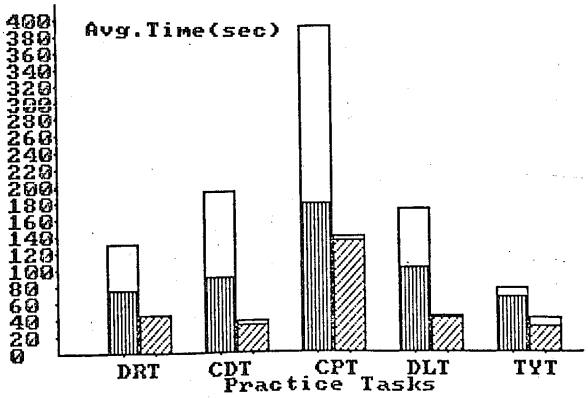

FIGURE 2. Practice Tasks With/Without Error Times

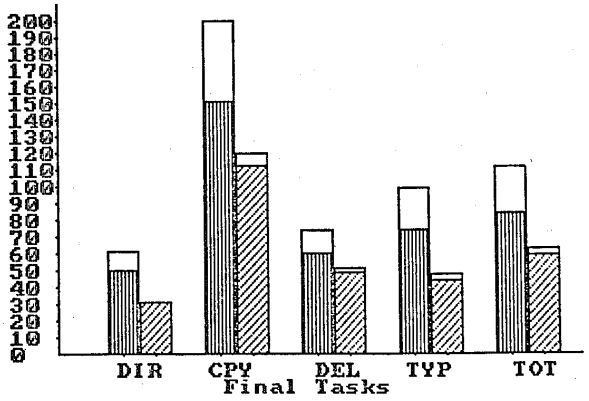

FIGURE 3. Final Tasks With/Without Error Times

DOS GPH

The results of this comparison are interesting in their own right. First of all there is a consistent performance advantage for the direct manipulation system. Further, this advantage was found to increase as the location of the file to be manipulated increased in depth in the directory hierarchy. For a 'worst case' scenario maximally favoring DOS (copy from the root directory to the root directory, renaming the file, and requiring a full directory scroll in the direct manipulation system), DOS subjects are faster than the direct manipulation subjects. However this was the only task (of the 32 general tasks and 24 practice tasks) in which this was the case.

4.1 Production Systems Models

Production system models were developed for the two systems used in the experiment. This was done to enable comparison of predictions of ease of learning and use from these models to be compared to those of the experimental data. While we do not employ the strict criteria for learning employed by Kieras and Polson in their laboratory work, there should be some correspondence between the performance times for our instructional phase and the number of productions in the two systems. While the situation in which our subjects learned the file management functions was not completely natural, it is closer to a real learning environment than that used in the Kieras and Polson work.

Guidelines for developing production system models were obtained from Polson and Kieras. These are 'style rules' which cover certain conventions which have been found useful in their previous work. The first two authors independently developed a production system for one of the systems. After this was done, the production systems were reviewed in a team (consisting of Karat, Fowler and Penny Smith-Kerker who is also working on production system models). Minor modifications were made based on these discussions. In general it took roughly one person weeks time to develop each production system.

Presenting a detailed description of the contents of the production systems would require an extensive discussion, and only some general features will be provided here. The two production systems were fairly similar in total size (i.e., number of productions). The GPH production system consisted of 49 productions, while the DOS production system consists of 43 productions. Both systems are fairly similar in structure, and both make the same sort of predictions concerning the relative difficulties of the functions (e.g. copy should be most complex, delete and type should be equally difficult). With the order of functions used in the experiment, there is assumed to be considerable transfer from the directory list function to the other functions (i.e., learning to specify a directory path in DOS, OR to point to a directory in GPH is assumed to transfer to the other functions in which this activity is a component). Basically, in terms of number of productions and transfer between function, the systems vary only a level of detail which is quite small.

If the instruction reading times from Figure 1 are taken as a measure of learning time (instruction reading time includes time to carry out sample tasks), then the number of new productions involved corresponds well to the copy, delete, and type function learning time data for both systems. For both DOS and GPH the reading time for the directory list function was far less than is predicted (it should be comparable to the COPY function time). This could be explained in a number of ways, and probably reflects a difficulty in assuring that a strict learning criteria has been reached by these subjects. Additionally, the significant difference in time between the DOS and GPH groups for the directory function is not predicted by the production system. From comments made by subjects during this phase, the difference would seem to be in the relative difficulty of understanding the path specification used in DOS compared to the spatial representation used in GPH. This difference did not show up in the production systems used.

Predictions of performance time are obtained from cycle counts by running the production system through the task set used by subjects. In the work of Kieras and Polson this is done by developing a GTN for the system involved, and having the simulated system and production system carry out the tasks. We did not actually develop a GTN for DOS and GPH, and ran the simulations by hand instead. The simulation suggested that compared to each other, the two systems would take similar times (within 10%) to carry out the functions. Both predict similar relative function difficulties, with the copy function approximately twice as long to execute as the other functions. Executing each of the other three functions involved similar numbers of cycles (i.e., list directory, delete file, and type file all involved similar production cycles).

Some aspects of these predictions are well reflected in the data (relative function difficulty within a system and transfer between functions), while others are not (comparison of difficulty between systems). Considering total time for each function for both practice and final tasks, the data show a significant advantage for GPH over DOS which is not predicted by the production cycle count. The difference is moderated (is either not significant or only marginally so) if time involved in error trials is excluded. There is no provision for errors in our production systems, and while one could be included, it would be a significant task involving the inclusion of problem solving mechanisms not well understood in cognitive psychology. This does not suggest that the production system analysis is wrong, only that it is incomplete in its current form.

5. DISCUSSION

The results of this study suggest that the direct manipulation file management system was much more easily used than DOS for the tasks involved. The differences were quite large for all but worst case scenarios for the direct manipulation system, and increase with level in the directory hierarchy. This result is consistent with the claims made for object oriented direct manipulation systems. However, these results were not anticipated by the production system analysis carried out on the systems involved. It is important to look at this failure in detail for causes and possible solutions (one of which may be to place limits on the usefulness of production system analysis of user interface complexity).

There were several issues in production system use that we intended on examining in this work. First, we wanted to find out if sufficient systems could be developed from the style rules provided by Polson and Kieras. Basically the result was mixed. While the task of writing production system models seems not terribly difficult (it was done by several members of our laboratory in time frames in the order of days rather than months or years), we all had a similar feeling of struggling with issues of granularity and uniqueness. While goal structures for tasks seemed to be easy to derive and agree upon, levels of analysis within the goals was not so easy. Take for example the question of how many productions should describe mouse pointing (what sort of level should one use to describe to homing behavior).

Some of these questions are less critical in a GOMS like model, where unequal components are acceptable, but when one is relying on 'all productions to be equal' for learning and performance predictions, the ability to maintain equal granularity across quite different environments is critical.

This may point to some limitations of the applicability of production system analysis. While Kieras and Polson have demonstrated that within environments involving limited differences good predictions can be derived, it may be that comparisons across quite different systems are not reliable. While behavioral studies could be used to help establish the'correct' number of productions for some given component (such as mouse pointing), the point of formal analysis is to try and bypass such analysis for engineering approximation.

The main failure in the predictions resulting from the production system model seems to lie in the inability to account for the large error times in the command language data compared to the direct manipulation data. This failure is not fundamental to the framework (i.e., it is easy to see how problem solving behavior could be 'built in' to the framework, even if the exact mechanisms are far in cognitive psychologies future), but it does suggest again that such considerations cannot be ignored in discussions of system usability. It would be nice if understanding and predicting expert behavior alone were sufficient, but it simply misses some very important facts.

This does not necessarily mean that the production system framework should be abandoned. We still need something beyond GOMS in order to address issues of ease of learning and transfer which are not easily addressed within that framework, and production system work has much promise if the difficulties can be overcome. But issues such as production granularity and the incorporation of problem solving mechanisms which are necessary for examining learning and transfer issues are not simple problems. In its current form, it does not appear that production systems provide a framework that would allow system designers to deal with these issues as well as the GOMS work deals with expert performance predictions.

What needs to be done to improve the usefulness of this approach takes the form of both extending the framework (to encompass areas such as error recovery behavior), and building a knowledge base.

Error behavior is simply too important to continue to ignore, and while there has been work in this area, it has not been incorporated in the Kieras and Polson framework. Additionally, there is considerable modularity in the production system framework which should be taken advantage of. Building a library of well understood models for common techniques (such as mouse pointing and keyboard error correction) would take much of the guess work out of model development. Finally, it would seem more useful to imbed the entire production system framework within a higher level task or goal oriented approach before expecting members of the software development community to find it useful.

REFERENCES

[1] 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.

[2] Newell, A., & Card, S. K. (1985). The prospects for psychological science in human-computer interaction. Human-Computer Interaction, 1, 209-242.

[3] Card, S. K., Moran, T. P., & Newell, A. (1983). The Psychology of Human Computer Interaction. Hillsdale, NJ: Erlbaum.

[4] Polson, P. G., & Kieras, D. E. (1985). A quantitative model of the learning and performance of text editing knowledge. Proceedings of the CHI 1985 Conference on Human Factors in Computing. San Francisco:ACM.

[5] Polson, P. G., Muncher, E., & Engelback, G. (1986). A test of the common elements theory of transfer. Proceedings of the CHI 1986 Conference on Human Factors in Computing. Boston: ACM.

[6] Ziegler, J. E., Hoppe, H. U., & Fahnrich, K. P. (1986). Learning and transfer for text editing with a direct manipulation interface. Proceedings of the CHI 1986 Conference on Human Factors in Computing. Boston: ACM.