THE CASE FOR HUMAN ENGINEERING

Henry Ledgard

Andrew Singer

COINS Tech. Report 77-11

September 1977

Computer and Information Science Department
University of Massachusetts, Amherst 01003

This work was supported by the National Science Foundation (Grant #DAAG-29-76-G-0216) and by the Army Research Office (Grant #MCS76-80854).

ABSTRACT

Typically, in computer systems, little attention is given to the design of the human interface. Consequently, systems are difficult to master and unpleasant to use. The continuing lack of an organized body of human factors knowledge that could guide system designers makes it unlikely that things will improve.

The basic contention of this paper is that much greater priority must be given to research in the human engineering of computer systems. The paper discusses the kinds of problems caused by poor human engineering, examines the resulting costs, and provides the beginnings of an annotated human factors bibliography for system designers.

"Mail could turn aside a sword blade but it could not keep out arrows and crossbow bolts. The armorers set out to find a new kind of armor that would. They began adding steel plates to the mail. For 150 years, as bows became more powerful and bowmen more skilled, they were forced to keep adding more plates: plates to plates, and plates on top of plates. By the time they had finished, they had the knight covered in plate armor from head to foot.

...

The armorers did not seem to know what they were doing at times. Many strange ideas were dreamed up and then thrown out. The knights themselves made things even worse than they might have been. They hated change of any kind. Many of them still hung on to their mail long after the others had changed. In England, they always seemed to be years behind.

...

When the knights began to wear plates, the longbowmen began to shoot at their horses; this was as good as killing the rider. When the knight was thrown from the saddle, he was usually stunned by the fall and ocassionally broke his neck. In any case, in sixty pounds of armor it was difficult for him to get back on his feet without help. French knights gave up riding into battle when they were fighting the English. They left their horses behind and walked. By the time they reached the English line, they were usually too tired to do any serious fighting. At the Battle of Agincourt in 1415, they had to tramp nearly a mile across rain-soaked fields to reach Henry V's tiny army. Five thousand French noblemen died in the mud that day and a thousand more were taken prisoner. Nobody bothered to count the number of common soldiers who fell. The English lost thirteen man-at-arms and about one hundred other men."

1. INTRODUCTION

In the development of any technology, there is always a tendency to lose sight of the basic problems that stimulated its introduction. After the first flush of success, interest naturally tends to focus on the technology itself and the problems it presents. So it was with armor in the fourteenth and fifteenth centuries. So it is with computers now. As armor evolved and armorers became more and more concerned with improving it, they forgot that the original purpose of armor was not just defense but to increase the overall effectiveness of the soldier. Today, we in computer science have forgotten that the original objective of computer technology was not just to develop more powerful systems but *to increase the overall effectiveness of a human problem solver.*

The literature shows that we are preoccupied with the technology rather than the users. Papers on the theory and design of programs, machines, languages, and algorithms abound. By contrast, a paper that addresses the human factors of computing is rare. In a similar vein, the recent spate of research in software quality has concentrated on programming technologies, control structures, management of large systems and reliability. All of these are aimed at producing better quality programs; yet none of them *directly* addresses the problem of producing programs that are easier to use. Surely, no one would argue that ease of use is not a factor in the "quality" of software.

Although meager, there has been some research directed at better human engineering of computer systems. This research has generally fallen into three categories: applications to specific designs, experiments on specific issues, and artificial intelligence.

In the first category, there have been several attempts to produce well human-engineered systems. A good example is the work of Wilcox, David, and Tindall [Wilcox 1976]. The authors have developed an on-line system that aids a user to find errors in source programs. The system allows a user to scan erroneous statements and obtain suggestions on possible errors. Furthermore, the authors make effective use of the PLATO terminal to give a user graphically annotated copies of program fragments. Unfortunately, it is not clear how realistic the given examples are, and the authors do not give any general principles for developing such aids.

In the second category, there are a number of experiments designed to test certain features of computer systems. An example is the experiment performed by Sackman [Sackman 1970]. This work describes an unusual experiment in which one subject, Sackman himself, used a self-tutoring manual for the TINT interpretive language. He had never used the system before and followed the self-tutoring manual completely for each lesson. During the experiment, he recorded his errors and documented his attitude toward the method. He observed that typing errors appeared at a constant rate, despite his knowledge of the system.

He also noted the tendency toward fatigue and boredom with the self-tutoring method, and suggested that complete reliance on self-tutoring is too extreme a method to use.

While we tend to agree with his conclusion, we are not really sure of what he set out to discover, and, as Sackman himself says, the study is "strictly exploratory and suggestive." Furthermore, we believe that such experiments should proceed from a thorough understanding of established results in the behavioral sciences. We strongly suspect that existing psychomotor studies on typing and the general research on learning could have helped him significantly in the design of the experiments and his interpretation of the results.

In the third category, there is work in artificial intelligence. Research in artificial intelligence is often directed at providing a very high level interface to the user. Typical topics include the development of adaptive systems and the use of natural languages. While these issues certainly can have a significant impact on the kind of human interface that computer systems provide, not enough thought is given to the constraints imposed by human limitations. Being able to build armor with flexible joints does not necessarily imply that one knows where the joints should be placed. What if, as a paper by Thomas and Gould [Thomas and Gould 1975] seems to suggest, natural language is not the best interface. How rapidly should systems adapt? When do the capabilities that a system provides exceed a person's ability to manage them? The answers to these questions depend on human, not machine, capabilities.

2. THE USER'S DILEMMA

The profound lack of concern for human factors in computing has conspired with the increased power of computer technology to aggravate rather than ameliorate the user. Like the development of plate armor, the development of systems has proceeded by accretion until the sheer complexity of systems has become a burden that users must bear.

Consider the following innocuous command:

RE[UM] [starting-number/10] [increment/starting number]

taken from a typical text-editor allowing line-numbers. Brackets denote optional fields. Thus, REN is an allowed abbreviation for RENUM. The first underline indicates that 10 is the default starting line number. The second underline indicates that the starting line number is the default increment between line numbers.

Let us begin by examining the abbreviation. In using an on-line system via the medium of a typewriter keyboard, abbreviations are critical to the rapid input of correct commands. On many systems there is no uniform abbreviation rule. Some keywords can be abbreviated by their first letter, others by two letters, still others by three, and some not at all. The user is thus forced to remember a detailed set of individual abbreviations. Our experience indicates that users respond to this situation in a variety of ways. Some make the effort to remember each abbreviation. Others just type commands out in full. Some try guessing at a general rule. Many bring their manuals to the terminal. None escape without paying in some way for the awkward abbreviation scheme.

Next consider the syntax of the command. A large number of psycholinguistic experiments seem to suggest that all aspects of human information processing are highly trained by natural language experience. We believe that interactive languages should be closely modeled after natural language phrase structure to take advantage of such training. A format like

RENUMBER [FROM starting-number] [INCREMENTING BY increment-number]

(where the individual phrases may appear in any order) or an entirely different approach like

START WITH number

INCREMENT BY number

RENUMBER

(where assumptions like START and INCREMENT are remembered by the system) takes advantage of established language habits. An appropriate general rule for either of these approaches might be to abbreviate any word by its first letter. Typically, command formats force the user to remember subtle, order-dependent fields. Such forms are error prone, distracting, and often conflict directly with the user's extensive natural language background.

Finally, there are a number of general issues that need to be resolved. Should a renumbering command be fully automatic, and not require user-specified options? Should text editors be based on line numbers at all? If so, what should the defaults be? Should the line numbers be considered part of the file during compilation?

There are all difficult problems. Yet in every system, there are a myriad of such minor "details." From the user's point of view these details are often the most frequently encountered consequences of the design. For example, the prompting signals given the user by a system are encountered in every interaction, while a highly specialized command may only be used once in a session, or perhaps not at all. It's the little things in life that count, because we experience them again and again. We believe that many of these apparently insignificant details are critical design issues, and may have far greater impact on users than the "large" design decisions.

Finally, Table 1 summarizes some typical problems with almost every system that we have encountered. Solving these problems is far from easy, but they are as pervasive as computer technology itself.

TABLE 1: The System's Armor

1. The language or system is too large. In an attempt to provide all things to all users, the language or system becomes so large that its complexity acts as a barrier to the user.

2. The documentation is incomprehensible. Most users of computer systems are not computer scientists by training, yet the documentation they must use is writen in the vocabulary of the computer expert.

3. The system provides no warning of a potentially dangerous action. It is often possible to destroy hours of valuable work through a single erroneous operation.

4. Language forms are difficult to remember. The syntax of each form has been individually designed to optimize some local goal, often minimal keystrokes. As a result, the system's language is unsystematic and thus difficult to learn and remember.

5. Abbreviation rules are not uniform. In an attempt to improve the ease of use, abbreviations are permitted. However, without a consistent rule, the saving in keystrokes is lost to the effort of remembering the right one. For example, how should the command to change a directory be written? Should it be CHADIR, CDIR, CHDIR, CHADR, or CDR?

6. Messages are often cryptic. Like the documentation, the system messages use the vocabulary of the computer expert and are often terse to the point of obscurity. While a short message may save printing time and be sufficient for the expert user, it offers the typical user little assistance.

7. Users are unable to communicate their difficulties to the designers. Many expert system designers soon forget their early experiences and frustrations. The systems they design show this, and without the feedback from typical users, there can be little hope for improvement.

8. Error creation is easy. Languages are designed with little consideration for the weaknesses of human information processing. For example, in some systems, users must be able to count commas visually; a task that few can perform reliably.

9. Error correction is difficult. Users, especially novices, inevitably make mistakes. The system often does little to help the user understand an error or to assist in its correction.

10. The system provides redundant forms of the same operation. Several methods of achieving exactly the same result are often provided. The user is thus left to search for subtle or nonexistent differences between methods.

11. The system requires that users learn much irrelevant information. In the attempt to do everything for all users, most users must learn about features that are of little relevance.

12. Assistance features are not an integral part of the system. While the need for user assistance has often been recognized, few truly usable automatic mechanisms have been developed.

13. Many useful tasks are not automated. Most systems provide elaborate facilities for file conversion, compilation, data base management, as well as libraries for sophisticated tasks. Yet common tasks like file preservation or the undoing of an erroneous action are seldom automated.

3. THE COSTS OF POOR HUMAN ENGINEERING

It is unfortunately true that the costs that result from a poorly human engineered system are more difficult to assess than the cost of the system itself. It is easy to measure central processor time, disk space, and input-output usage. Nevertheless, while human costs are less obvious, they may be staggering.

We see four areas where the costs of poor human engineering are evident: *direct costs, human suffering, indirect costs,* and *limits to use.* We believe these areas are in roughly increasing order by cost, with the last by far the most expensive.

Direct costs are easy to point out. Poor designs often lead to wasted human time at a terminal, as well as wasted computer time. How often have we seen a user receive an unusual but cryptic message, and try over and over again to repeat a sequence of commands in an attempt to skirt the message? The simple failure to save a file may cost hours. And what of the excessive file duplication to which users resort to prevent such inadvertent losses? All of this wastes time and resources.

Of course, we could recount some of the nightmares we have heard in recent years. We have all paid our share of direct costs. How many of us remember an aborted program run, searching through core dumps (the nadir of human engineering), retrials, rethinking, searching for system bugs, and more trials, just because of a simple mistake? How nice it would have been to be warned of the possibility of error in the first place.

The second area, which we call *human suffering*, is more subtle. By this we mean the class of negative psychological effects associated with the use of a system. The frustration that results from repeated mistakes due to some error-prone feature of a system is one example of the misery that systems impose on people. Another is the anxiety that results when a system seems to behave erratically. Yet another is the pressure and fatigue caused by frequent difficulties in understanding system responses, dealing with irregular behavior, and keeping track of mentally difficult exercises that systems demand.

Of course, there is no reasonable way one can put any cost estimates on these kinds of problems. Nonetheless, they are some of the worst consequences of our profession, and we pay for them. In some instances, these costs may be so great that no individual can really evaluate them.

In the third area, we lump a large number of *indirect costs*. By "indirect" we mean, "not associated with direct machine usage," but rather with the day-to-day problems on the periphery of man-machine encounters.

The vast scale of most systems presents a ready example. What price to we pay for trying to understand volumes of system documentation? Learning the many system commands and options? Attempting to understand the large number of system-generated messages? Mastering a truly effective sequence of commands to perform some simple operation?

There are perhaps hundreds of indirect, cost-related issues. We list but a few:

... the time spent in understanding the vocabulary of a system,

... the cost of printing excessive documentation,

... the time spent in understanding manuals,

... the time spent in learning abbreviations,

... the time spent in mastering system formats,

... the lateness of projects caused by having to deal with a difficult system,

... the lack in communication of ideas among users,

... the time spent in understanding errors,

... the mental overhead in remembering duplicate forms,

... the manual costs of features that should have been automated.

We need not go on.

In the last area, *limits to use*, we come to what we believe is the major cost of poor human engineering. Indeed, all of the foregoing problems can only force us to conclude that:

- *There are many people who should, but do not, use computers.*

How many scientists use a computer in their day-to-day calculations? How many managers regularly use computers to process day-to-day information? How many educated computer scientists use a text editor on a daily basis? What if the user is a secretary? An engineer? Or Linus Pauling? Or you?

If we do not attack problems people face in using computers, our systems will ultimately become lonely citadels, effectively armed against those who need them most. Then, like the French at Agincourt in 1415, we will discover the consequence of a technology that has taken us too far in the wrong direction.

4. READINGS IN HUMAN ENGINEERING

We have been attempting to put together a list of readings in human engineering. At present, these readings are grouped into five categories:

1. Motivation: These papers include words describing problems in human engineering, and works that motivate a concern for better human factors in computer systems.

2. Surveys and Bibliographies: This group comprises several related attempts to survey work on some area relevant to our goal.

3. Application to Computer Systems: These papers generally describe systems where some form of human engineering was important in the design.

4. Results from the Computer Sciences: The papers in this category generally give statistical or experimental results on some aspects of computer design.

5. Results from Behavioral Sciences: Papers in this category describe work that we believe to be of relevance to the design of computer systems.

Of some controversy is the collection of papers in the last category, which represent the results of a small initial investigation. Although we strongly believe that there is much work in the behavioral sciences that would be of value to system designers, we are not yet prepared to make the case for including this literature. Consequently, we have deliberately excluded such papers from the bibliography that follows.

ACKNOWLEDGMENTS

We are grateful to James Carlisle and John Gould for their helpful critiques of our work on this paper.

MOTIVATION

[Bailey et al. 1973]

Robert W. Bailey, Stephen T. Demers, and Allen I. Lebowitz
Human Reliability in Computer-Based Business Information Systems
IEEE TRANSACTIONS ON RELIABILITY Volume R-22(3), August 1973, p. 140-148

This paper identifies factors that could increase the probability of error in a computer-based information system. These factors are given a rating of their potential order of effect and grouped into categories, such as personal considerations, documentation, training, man-machine interface, and environmental issues. Little supporting evidence is offered for the ratings, but the article provides a list of potential problem areas.

Keywords: Error proneness, design criteria

[Caldwell 1975]

John Caldwell
The Effective Reports Crisis
JOURNAL OF SYSTEMS MANAGEMENT, June 1975, p. 7-12

This is an informal position paper making the observation that despite its widespread creation and use, hard copy output is often badly human engineered.

Keywords: Reports, hard copy output

[Conrad 1967]

R. Conrad
Designing Postal Codes for Public Use
THE HUMAN OPERATOR IN COMPLEX SYSTEMS, W.T. Singleton et al. (eds.), Taylor and Francis Ltd., London, 1967, p. 132-138

This paper demonstrates the application of experimental results to the human engineering of codes. The author discusses the problems of designing a usable postal code and presents a sample code.

Keywords: Codes, design methodology

[Cooke and Bunt 1975]

John E. Cooke, and Richard B. Bunt
Human Error in Programming: The Need to Study the Individual Programmer
Dept. of Computational Science Technical Report 75-3, University of Saskatchewan, Canada, 1975

This paper calls for a greater attention to the study of individual programmers. Among other points, the authors claim that little attention has been paid to the human memory and problem solving skills needed for programming, and that there is a general lack of research on human performance in computer systems.

Keywords: Programming, problem solving

[Cuadra 1971]

Carlos A. Cuadra
On-Line Systems: Promise and Pitfalls
JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE, March-April 1971

This paper makes a strong case for better human engineering of information retrieval systems. The author claims that we do not need new technological breakthroughs to exploit the potential of on-line systems, but we do need breakthroughs in organizing for technological change.

Keywords: Information systems, use of technology, social implications

[Haynes 1977]

James Haynes
A Tale of Two Computers
COMPUTER, IEEE Computer Society, May 1977

This paper is intended as a moral lesson on the price people pay for poor designs. It gives a short "horror" story of a painful encounter with a computer system. The story illustrates that a lack of human engineering can sometimes lead to extreme problems.

Keywords: Error proneness, design criteria

[Holt and Stevenson 1977]

H. O. Holt, and F. L. Stevenson'
Human Performance Considerations in Complex Systems
SCIENCE, Volume 195, 1977, p. 1205-1209

This paper discusses the early work on human engineering of physical devices and the need for similar work in computer systems. The authors contend that good human engineering must be a fundamental consideration during hardware and software design and that it cannot be added at the end of the design process.

Keywords: Design, design methodology

[Kennedy 1974]

T.C.S. Kennedy
The Design of Interactive Procedures for Man-Machine Communication
INTERNATIONAL JOURNAL OF MAN-MACHINE STUDIES, Volume 5, 1974, p. 309-334

This article calls for better human engineering of man-machine systems. The author proposes that humans should be able to communicate in a terse "natural" language, that humans should be able to control the amount of interaction, that error correction should be easy, and that a computer system should be able to give assistance whenever it is needed.

Keywords: Natural language, design criteria

[Mann 1975]

William C. Mann
Why Things are So Bad for the Computer-Naive User
PROCEEDINGS OF THE NATIONAL COMPUTER CONFERENCE, 1975,
p. 785-787

This short article presents some of the problems naive users face in dealing with computers. It points out that research in man-machine communication deserves a far higher national priority.

Keywords: Naive users, social implications

[Palme 1975]

Jacob Palme
Interactive Software for Humans
Research Institute of National Defense, Stockholm, Sweden,
NTIS No. PB-245 553, July 1975

This paper presents a set of arguments calling for better human engineering of computer systems. The author draws many of his ideas from the view that man-machine communication should be similar to man-man communication.

Keywords: Man-machine interface, social implications, interactive systems, natural language

[Parsons 1970]

Henry M. Parsons
The Scope of Human Factors in Computer-Based Data Processing Systems
HUMAN FACTORS, Volume 12(2), 1970, p. 165-175

This paper is a prosaic discussion of the importance of human factors in the computer field.

Keywords: Social implications, displays, hard copy output

[Sterling 1974]

Theodor D. Sterling
Guidelines for Humanizing Computerized Information Systems: A Report from Stanley House
COMMUNICATIONS OF THE ACM, Volume 17(11), November 1974

This paper presents a strong case for better human engineering and offers guidelines for improved designs. The author claims that the effects of computers on humans are subtle but pervasive, and that the utility of human engineering arises from the point of view of the quality of life. Among the guidelines offered, the author proposes that transactions with a system should be courteous, that there should be provisions for users to correct errors, and that systems should recognize different classes of users.

Keywords: Social implications, design criteria

[Vandenberg 1967]

J.D. Vandenberg
Improved Operating Procedures Manuals
ERGONOMICS, Volume 10(2), 1967, p. 114-120

This paper presents a list of general recommendations in designing operating manuals for physical tasks. Some recommendations are:

(1) provide several means of indexing into information,
(2) keep the description of one item in one place,
(3) keep references to other documents to a minimum,
(4) make all cross-references specific (e.g., give page numbers),
(5) describe the response to an incorrectly used operation

Its recommendations appear to be applicable to computer manuals.

Keywords: Documents, manuals, design criteria

[Weinberg 1971]

Gerald Weinberg
THE PSYCHOLOGY OF COMPUTER PROGRAMMING
Van Nostrand Reinhold Co., 1971

This book is one of the first works to have a major impact on psychological concerns in computer science. Topics include the organization of programmer groups, social factors, language design, and characteristics of good programs. The major theme of the book is the need for treating programming as a human process.

Keywords: Programming, programming groups, programming languages, psychological aspects of programming

SURVEYS AND BIBLIOGRAPHIES

[Gould et al. 1971]

John D. Gould, Walter J. Doherty, and Stephen J. Boies
Bibliography of Behavioral Aspects of On-Line Computer Programming
IBM Watson Research Center, Technical Report RC-3513, Yorktown Heights, NY, August 1971

This bibliography references a number of articles published before 1971. The bibliography is broken into two parts: (1) studies containing data, and (2) studies not containing data.

Keywords: Bibliographies, experimental data

[Riddle 1976]

Elizabeth A. Riddle
A Comparative Study of Various Text Editors and Formatting Systems
Project 0147A, No. AD-A029050, Defense Documentation Center, Cameron Station, Alexandria, VA, August 1976

This is a detailed survey paper on several existing text editors and text formatting systems. The author proposes some desirable features for editors used for file management, program editing, and document preparation. Some of the recommended features are:

(1) the ability to deal with structural features of a file,
(2) provision for both line and string editing abilities,
(3) forward and backward searching facilities,
(4) a comprehensive assistance facility,
(5) allowances for multiple commands on a single line,
(6) inclusion of programmable features (executable macros).

Two lengthy appendices compare nine current editing systems on a feature-by-feature basis.

Keywords: Surveys, text editors, terminals, design criteria

[van Dam and Rice 1971]

Andries van Dam, and David E. Rice
On-Line Text Editing: A Survey
COMPUTING SURVEYS, Volume 3(3), September 1971

This paper surveys several major text manipulation systems. The paper discusses the advantages of on-line editing systems, design goals, and the characteristics of several systems.

Keywords: Surveys, text editors, terminals

[... 1974]

Man-Machine Interaction. Bibliography December 1953 - March 1972
Defense Documentation Center, Alexandria, VA
Abstract in ERGONOMICS 17(1), January 1974, p. 119

This is a rather lengthy bibliography of early papers on man-machine interaction. No attempt is made to evaluate the material.

Keywords: Bibliographies

[Anderson and Gillogly 1976]

R. H. Anderson, and J. J. Gillogly
Rand Intelligent Terminal Agent (RITA): Design Philosophy
No. R-1809-ARPA, Rand Corporation, Santa Monica, CA, 1975

This paper describes an intelligent terminal facility for general use of the ARPANET. The terminal programs and user language are centered on the method of "production systems." The paper presents no clear idea of what the system as a whole does, for the paper is more about work in progress.

Keywords: Intelligent terminals, design criteria

[Gilb and Weinberg 1977]

Thomas Gilb, and Gerald M. Weinberg
HUMANIZED INPUT: TECHNIQUES FOR RELIABLE KEYED INPUT
Winthrop Publishers, Inc., Cambridge, MA, 1977

This work considers a common, but seldom discussed, problem: the design of keyed input. The authors make a convincing case that this area has generally been neglected, propose a design philosophy, and give a number of design strategies for making computer input easier and more reliable. In particular, the authors consider topics like the use of default values, differences between positional and identified items, and the avoidance of error-prone forms for data entry. The work is supported by many examples.

Keywords: Keyed input, error proneness, design criteria

[Grignetti et al. 1975]

M. Grignetti, J. Gould, and C. Hausmann, et al.
NLS-Scholar: Modifications and Field Testing
ESD-TR-75-358, Bolt, Bernaek, and Newman, Inc., Cambridge, MA 1975

This paper describes an interactive tutorial system for helping naive users learn about a text editor. The entire dialogue language is based on a fragment of natural English. The system gives the user the option to suspend what he is doing in order to ask questions, practice, etc. The system was apparently designed with a deep concern for human engineering.

Keywords: Assistance facilities, text editors, learning, design criteria

[Johnson 1967]

E. A. Johnson
Touch Displays: A Programmed Man-Machine Interface
ERGONOMICS, Volume 10(2), 1967, p. 171-177

This paper makes a case for touch displays. It is claimed that such displays provide an effective means for user-terminal interaction. The case made is moderately convincing but it remains to assess the total impact of using such a device is yet to be assessed.

Keywords: Displays, terminals, keyboards

[Singer et al. 1976]

Andrew Singer, Jon Hueras, and Henry Ledgard
A User's Guide to the PASCAL Assistant
Technical Report, Computer and Information Science,
University of Massachusetts, Amherst, June 1976

This report describes a proposed system with a simple user interface that hides a complex programming facility. The proposed system is claimed to be based on human engineering principles. Among other features, the user language is based on a limited syntax modeled after natural language, the user has the capability of undoing a command, and checks are made before a potentially dangerous operation.

Keywords: Design criteria, scale, natural language model

[Stewart 1976]

T.F.M. Stewart
Displays and the Software Interface
APPLIED ERGONOMICS, Volume 7(3), 1976, p. 137-146

This paper is a recapitulation of ideas for designing computer display facilities. For example, the author considers input sequences, output format, error detection, and display size. The ideas are supported by reasoned arguments and further references are provided.

Keywords: Displays, terminals, error detection, perception

[Wilcox et al. 1976]

Thomas Wilcox, Alan M. Davis, and Michael H. Tindall
The Design and Implementation of a Table Driven, Interactive Diagnostic Programming System
COMMUNICATIONS OF THE ACM, Volume 19(11), 1976, p. 609-616

This paper describes a system that aids programmers in finding errors in source programs. Among other features, the system allows the user to scan through erroneous statements and obtain suggestions on possible errors. Furthermore, the authors make effective use of the PLATO terminal in giving the user good graphically annotated copies of program fragments containing errors. While the authors do not present any general principles for developing such aids, the system was apparently designed with a strong concern for the user.

Keywords: Program testing, assistance facilities, error correction, terminals

[Wright and Barnard 1975]

P. Wright and P. Barnard
Just Fill in this Form - A Review for Designers
APPLIED ERGONOMICS, Volume 6(4), 1975, p. 213-220

This article gives a summary of work on the design of forms. Among the principles discussed are:

(1) Use short, active, affirmative sentences.
(2) Use familiar words and avoid ambiguities.
(3) Ask one question at a time and arrange the questions in temporal sequence.
(4) Consider alternatives to prose.

The article contains a comprehensive bibliography and is supported by some well chosen examples.

Keywords: Forms, design criteria

RESULTS FROM THE COMPUTER SCIENCES

[Boies 1974]

Stephen J. Boies
User Behavior on an Interactive Computer System
IBM SYSTEMS JOURNAL, Number 1, 1974, p. 2-18

This paper describes a statistical study of user behavior on a time-sharing system. The study shows that in a research center environment many programs were compiled or assembled error free, and that text editing commands were the most frequently used. The study also showed that many users made only occasional use of the system.

Keywords: Statistics, interactive systems, text editors

[Boies and Gould 1974]

Stephen J. Boies, and J. D. Gould
Syntactic Errors in Computer Programming
HUMAN FACTORS, Volume 16(3), 1974, p. 253-257

This paper describes the study in [Boies 1974] and points out that syntactic errors were not a major obstacle for professional programmers, and that automatic error correction facilities were seldom used.

Keywords: Statistics, error correction, syntax errors

[Gannon 1977]

John D. Gannon
An Experimental Evaluation of Data Type conventions
COMMUNICATIONS OF THE ACM, Volume 20(8), August 1977,
p. 584-595

This paper describes an experiment to test the supposed reliability of statically typed languages over languages without type distinctions. The author devised two languages, one with two statically distinguished types (integers and strings) and one with a single type (words containing characters). Subjects were asked to write programs in both of the languages. The results showed a significantly fewer errors with use of the typed language.

Keywords: Experiments, programming languages, error proneness, data types

[Gannon and Horning 1975]

John D. Gannon, and James J. Horning
Language Design for Programming Reliability
IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, Volume SE-1(2),
1975, p. 179-191

This paper describes an experiment on language design. The authors used an existing language and constructed a variant using design principles believed to be less error-prone. In particular, the languages differed on issues like the use of semi-colons at the end of program statements, block structure, and precedence rules. Students were asked to write programs in either the initial language or in its variant. The results demonstrated that changes in these areas can lead to fewer errors.

Keywords: Experiments, programming languages, error proneness

[Gould and Drongowski 1974]

John D. Gould, and Paul Drongowski
An Exploratory Study of Computer Program Debugging
HUMAN FACTORS, Volume 16(3), 1974, p. 258-277

This paper describes an experiment devoted to various debugging ideas. Programmers were asked to correct twelve programs with one semantic error each. They were told that the programs contained one error and were given one of five items:

(1) a program listing,
(2) a program listing plus sample data and results
(3) a program listing, sample data, results, and the correct results,
(4) a program listing plus the kind of error in the program, or
(5) a program listing and the number of the line containing the error.

The results showed a clear advantage in knowing the line number of an error. It also showed that having just the listing was as useful as having the listing plus the class of error or a sample input-output.

Keywords: Experiments, program testing, error correction

[Grossberg et al. 1976]

Mitchell Grossberg, Raymond A. Wiesen, and Douive B. Yntema
An Experiment on Problem Solving with Delayed Computer Responses
IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS, March 1976, p. 219-222

This paper reports an experiment intended to measure the effects of unpredictable delay in giving responses to an interactive problem solver. Previous studies had suggested that long delays were distracting to users and disproportionately increased task completion times. The experiment tends to disprove previous results. The experiment appears to suffer from several methodological errors, for example, the lack of experimental controls.

Keywords: Experiments, interactive systems, problem solving

[Love 1977]

Thomas Love
An Experimental Investigation of the Effect of Program Structure on Program Understanding
SIGPLAN Notices, Volume 12(3), March 1977, p. 105-113

This paper describes two experiments intended to determine whether the choice of control structures and the use of prettyprinting (paragraphing and indentation) affect program understanding. Understanding was measured by asking subjects to recall a program verbation. The results indicated some positive effect when using simplified control structures and no significant effect when paragraphing was used. Unfortunately, this paper suffers from several methological problems. These include the use of tiny programs and the use of verbation recall to measure the borad effects of control structures and paragraphing.

[Miller 1974]

Lance A. Miller
Programming by Non-Programmers
INT. JOURNAL OF MAN-MACHINE STUDIES, Volume 6, 1974, p. 236-260

This paper discusses an experiment using nonprogrammers to write simple algorithms in a simple language. The author made an analysis of programming errors and the structure of the "programs." The results tended to confirm the concept-learning difficulties associated with disjunction and negation. A main point claimed by the author is the feasibility of using naive subjects to investigate programming.

Keywords: Experiments, language, naive users, concept learning

[Miller and Becker 1974]

Lance A. Miller, and Curtis A. Becker
Programming In Natural English
IBM Research, RC5137, Yorktown Heights, NY, 1974

This paper describes the authors' philosophy for studying programming via natural language. The paper also makes a case for studying how lay persons program computers. The bulk of the paper discusses an experiment in which 14 lay persons were asked to supply English procedural solutions to six problems. The results showed that subjects used diverse terminology and methods. It also suggests that the strict discipline (e.g. logic, flow of control, and precise specification) required for programming is an unfamiliar approach for humans.

Keywords: Experiments, natural language, naive users

[Miller 1975]

Lance A. Miller
Naive Programmer Problems with Specification of Transfer-of-Control
Proceedings of National Computer Conference, AFIPS Volume 44, 1975, p. 657-663

The paper argues that naive users have difficulty in dealing with the control structures of conventional programming languages. Some other control structures are suggested, but no data are given to support the conclusions.

Keywords: Programming languages, naive users, control structures

[Miller 1976]

Laurence H. Miller
An Investigation of the Effects of Output Variability and Output Bandwidth on User Performance in an Interactive Computer System
ISI/RR-76-50, ARPA Order No. 2223, Information Sciences Institute, Marina Del Rey, CA, December 1976

This paper examines the effect of the display rate of a terminal on user performance. The study suggests that increasing the display rate does not necessarily improve user performance. It further suggests that greater display variability (characters appearing at uneven display rates) degrades performance.

Keywords: Experiments, interactive systems, terminals, response time

[Nagy and Pennebaker 1974]

G. Nagy, and M. Carlson Pennebaker
A Step Toward Automatic Analysis of Student Programming Errors in a Batch Environment
INTERNATIONAL JOURNAL OF MAN-MACHINE STUDIES, Volume 6, 1974, p. 563-578

This paper examines the changes made to some relatively simple PL/I programs written by novice programmers. The authors collected data using an automated system. The authors noted that most changes occurred in declarations, followed by conditional, WRITE, and assignment statements. While recognizing that automated systems can yield only limited amounts of information, the authors make a strong case for automatic collection of such data.

Keywords: Statistics, error proneness, PL/I, programming languages

[Sackman 1970]

Harold Sackman
Time-Sharing and Self-Tutoring: An Exploratory Case History
HUMAN FACTORS, Volume 12(2), 1970, p. 203-214

This paper describes an unusual experiment in which one subject (the author) used a self-tutoring manual for the TINT interpretive language. The subject had never used the system before and followed the self-tutoring manual through each lesson. During the experiment, errors were recorded and the subject documented his attitude during the entire experiment. The author noted that typing errors appeared at a constant rate, despite his knowledge of the system. He also noted the tendency toward fatigue and boredom with the self-tutoring method and pointed out complete reliance on self-tutoring is a drawback.

Keywords: Experiments, learning, interactive systems

[Shneiderman and Mayer 1975]

Benjamin Shneiderman, and R. Mayer
Towards a Cognitive Model of Programmer Behavior
Technical Report 37, Indiana University, Bloomington, 1975

This paper discusses a tentative model of programmer behavior.

Keywords: Programming, models, problem solving

[Shneiderman et al. 1977]

Benjamin Shneiderman, R. Mayer, D. McKay , and P. Heller
Experimental Investigations of the Utility of Detailed Flowcharts in Programming
COMMUNICATIONS OF THE ACM, June 1977, p. 373-381

This paper treats one of the most widely held notions in programming: the utility of flowcharts. The paper discusses some experiments regarding program comprehension and program modifications with and without the use of flowcharts. The results indicate no significant improvement when flowcharts were used.

Keywords: Experiments, flowcharts, programming

[Thomas 1976a]

John C. Thomas
Quantifiers and Question-Asking
IBM Thomas J. Watson Research Center Technical Report RC-5866
Yorktown Heights, NY, 1976

This paper discusses the use of quantifiers in languages for man-machine communication. Although no firm conclusions are drawn in the paper, it makes a case that quantifiers like "all" and "some" can be difficult for users to handle.

Keywords: Experiments, language

[Thomas 1976b]

John C. Thomas
A Method for Studying Natural Language Dialogue
IBM Thomas J. Watson Research Center Technical Report RC-5882, Yorktown Heights, NY, 1976

This paper discusses some techniques for studying how users would communicate with a computer if natural languages were available. The author makes a case for natural language communication and points out that users with varying experience communicate differently.

Keywords: Natural language, interactive languages

[Thomas and Gould 1975]

John C. Thomas, and John D. Gould
A Psychological Study of Query by Example
Proceedings of the National Computer Conference, AFIPS, Volume 44, 1975, p. 439-445

This paper demonstrates the power of a specialized, very high-level language for interactive use. The authors conducted several experiments on the ability of nonprogrammers to learn and use Zloof's Query By Example language. This language was compared with several competing systems. The authors appear to be very careful in their experimental design, and the results confirm a superior performance for most users. The paper demonstrates by example that specialized notations, particularly a familiar one like charts and examples, may at times be superior to natural language.

Keywords: Experiments, information systems, specialized systems

[Weissman 1974]

Laurence M. Weissman
A Methodology for Studying the Psychological Complexity of Computer Programs
Computer Systems Research Group, University of Toronto
Technical Report CSRG-37, Canada, 1974

This report proposes a methodology for testing the suitability of properties of programs and describes a series of experiments based on the methodology. The methodology consists of choosing a property of programs, constructing versions of a program to illustrate the use and nonuse of the property, and testing subjects on the various versions. The experiments tested properties based on current ideas in software quality, e.g. use of paragraphing, use of mnemonic names, and locality of data references. The results tended to confirm current ideas in software quality.

Keywords: Experiments, methodology, programming, programming languages