Design Rules Based on Analyses of Human Error

DONALD A. NORMAN University of California. San Diego

Donald A. Norman's research covers "cognitive engineering," which provides design tools for the development of interactive systems, for both novices and experts, which might reduce the incidence of human error. He is one of the founders of the Cognitive Science Society and is an editor of Cognitive Science.

Research support was provided by Contract N00014-79-C-0323, NR 667-437 with the Personnel end Training Research Programs of the Office of Naval Research. sponsored by the Office of Naval Research and the Air Force Office of Scientific Research. Requests for reprints should be sentto the author.

Author's Present Address Donald A. Norman. Program in Cognitive Science C-015; University of California, San Diego: La Jolla, California, 92093, USA.

ABSTRACT: By analyzing the classes of errors that people make with systems, it is possible to develop principles of system design that minimize both the occurrence of error and the effects. This paper demonstrates some of these principles through tbe analysis of one class of errors: slips of action. Slips are defined to be situations in which the user's intention was proper, but the results did not conform to that intention. Many properties of existing systems are conducive to slips; from tbe classification of these errors, some procedures to minimize the occurrence of slips are developed.

INTRODUCTION

It should come as no surprise to learn that people make errors in the use of computer systems. Experienced users take this as a matter of course, often smiling tolerantly as new users repeat well-known errors. Errors, however, can be serious, both in the sense that they can lead to serious mishap, and in the social sense that they frustrate beginning users. There is little need for most of these errors. Most are system induced, a result of inappropriate system design. In part, these errors spring from insensitivity on the part of the designer to the needs and functions of users of all abilities. from novice to expert (and, despite their protests, themselves). This paper addresses the development of design rules that can lead to improvements in this situation. Today, proper tools for design do not exist. Thus, the designer who wishes to minimize error has no standard reference to turn to for advice. This paper makes a modest start in the direction of providing appropriate rules.

SYSTEM DESIGN PRINCIPLES CAN BE DERIVED FROM THE CLASSES OF HUMAN ERROR

I have studied the errors made by people in a variety of situations, including the use of computer systems [8, 9]. These errors yield some insight into the psychological mechanisms that are involved, and they can also be used to examine the human-machine interface.

Call the highest level specification of a desired action an intention. The intention may result from conscious decision making or from subconscious processing. The important point is that it is a high level specification that starts a chain of processing that normally results in the accomplishment of that intention. An error in the intention is called a mistake. An error in carrying out the intention is called a *slip*. Both classes of errors are important; each gives rise to different forms of difficulty, with different underlying principles and suggested solutions. Here, I concentrate upon an analysis of slips.

Slips can be classified into a small set of classes based upon the mechanisms that seem to be the most likely causes. The basic classification is based upon a simple

model of the human in which it is assumed that any intention causes a number of schemas to become active, each with an activation value and a set of trigger conditions (see [8], for details); a schema controls behavior whenever the combination of its activation value and the goodness of match of its trigger conditions reaches an appropriate level. This model gives rise to the classification of slips listed in Table I. Preventable errors in human-computer interaction come primarily from a subset of the classification scheme shown in Table I. in particular, from mode, description, capture, and activation errors.

MODE ERRORS SUGGEST THE NEED FOR BETTER FEEDBACK

Modes will be with us as long as we wish to be able to do more operations than we have special keys. Most complex devices end up having special keys, be they digital watches, aircraft automatic pilots, or text editors. A large class of errors is the *mode error* : doing the operation sppropriate for one mode when in fact you are in another. Mode errors occur when the person believes the system is in one state (mode), when it is actually in another. This leads to the performance of an inappropriate action. Mode errors occur frequently in systems that do not provide clear feedback of their current state.

The most common examples in my collection come from the use of computer text editors, where users try to issue commands while in text mode or to type text while in command mode. Similar errors occur in pushing the buttons on complex digital watches. The autopilots of commercial aircraft provide numerous possibilities for mode errors; a recent incident in which an Aero Mexico DC-10 stalled, was badly buffeted, and lost the tips of both elevators appears to have occurred because of a mode error in using the autopilot on the part of the crew [6]. The clear implication of mode errors is that they result from inadequate feedback and indication of the state of the system. Many mode errors occur in text editors. with confusions between "insert" mode and *command" mode. In any event, there are obvious ways to minimize mode errors.

- Do not have modes.
- Make sure that the modes are distinctively marked.
- Make the commands required by different modes different, so that a command in the wrong mode will not lead to difficulty.

These suggestions cannot always be followed, and if they are, they can sometimes lead to yet other classes of difficulties. The designer must balance the tradeoffs associated with one procedure against those of another.

In order to identify the system state or mode, some systems have rows of lights. Such indicators are clearly well intentioned, but they lead to another class of errors: **description errors.** (Designers who make pretty panels by having a row of identical switches or displays invite errors in which you do the right operation on the wrong item. The middle lights of a row of lights are easily confused, especially when the user is rushed or when they are viewed with peripheral vision.) It is possible to minimize the number of modes in a system by appropriate design procedures, for example, the use of menus and pointing devices. However, for complex systems, I do not believe it is possible to eliminate modes, and, moreover, the partial elimination usually creates new problems for users-usually a penalty for the expert user. Indeed, elimination of modes by the creation of separate switches or commands for each desired operation can simply trade one class of errors for another: mode errors for description errors.

TABLE I. A Classification of Slips Based on Their Presumed Sources.[a]

Slips in the **Formation of Intention**
- **Mode errors:** Erroneous classification of the situation
- **Description errors: Ambiguous or incomplete specification of the intention**

Slips Resulting from Faulty Activation of Schemas
- *Unintentional* **activation: When schemas not part of a current action sequence become activated for extraneous reasons. then become triggered and lead to slips**
- *Capture* **errors: When a sequence being performed is similar to another more frequent or better learned sequence. the latter may capture control**
- **Dotadrrven activation: External events cause activation of schemas**
- *Associative activation*: **Currently active schemas activate others with which they are associated**
- **Loss of *activation*: When schemas that have been activated lose activation. thereby losing effectiveness to control behavior**
- *Forgetting* **an *intention* (but continuing with the action sequence)**
- *Misordering* **the components of an action sequence. including skipping steps and repeating steps**

Slips Resulting from Faulty Triggering of Schemas
- *False triggering*: **A properly activated schema is triggered at an inappropriate time**
- **Spoonerisms: Reversal of event components**
- **Blends: Combinations of components from two competing schemas**
- *Thoughts* **leading to actions: Triggering of schemas meant only to be thought. not to govern action**
- *Premature triggering*
- **Failure *intriggering*: When an active schema never gets invoked because**
  - **Action was preempted by competing schemas.**
  - **There was insufficient activation. either as a result of forgetting or because the initial level was too low.**
  - **There was a failure of the trigger condition to match, either because the triggering conditions were badly specified or the match between occurring conditions and the required conditions were never sufficiently close.**

[a] Modified from [8]

DESCRIPTION ERRORS SUGGEST THE NEED FOR BETTER SYSTEM CONFIGURATION

A description error occurs when there is insufficient specification of an action and the resultant ambiguity leads to an erroneous act. Usually this erroneous act is closely related to the desired one. Often the errors are humorous (at least to others). One dramatic case occurred when a person, cleaning a fish in a rowboat in the middle of a lake, threw the cleaned fish overboard and kept the entrails. In another case, a person preparing for a party put the cake in the refrigerator and the salad in the oven. Description errors also occur in operational situations, where they can lead to serious accidents [7]. The two preceding description errors can be thought of as situations in which the proper arguments and functions were specified, but the ordering of the arguments was improper. In general, these errors occur when different actions have similar descriptions, either in the specification of the actions or in the class of arguments.

One class of description errors occurs in the use of computer text editors which have multiple commands, usually based upon one or two keystrokes. Thus. in the text editor "vi" (the screen editor supplied with the Berkeley Distribution of the UNIX operating system). each of the letters d, f, g, and u has a different meaning when typed in lower case ("d"), upper case ("*shift-d" or "D"), or as a control key ("control-d"). Many other keys also have these multiple uses. It should come as no surprise to discover that description errors occur frequently in this editor.

Description errors are relatively common in the throwing of switches or operations of controls, especially when the operations are similar, such as in the setting of altimeters. radio frequencies. and transponder codes. This problem is especially bad in the design of nuclear power plant control rooms, where switches and controls are laid out in neat, logical, nice-looking rows. The result, however, is clear potential for confusion, for reading the wrong instruments. and for operating the wrong controls [5].

Description errors can be expected to occur wherever control panels are designed so that at a quick glance (or in peripheral vision) the distinctions among controls are not clear enough. Solutions to this problem have long been known:

1. Arrange instruments and controls in functional patterns. perhaps in the form of a flow chart of the system.
2. Use "shape coding" to make the controls and instruments look and feel different from one another.
3. Make it difficult to do actions that can lead to operations that have serious implications and that are not reversible.

With computer systems, these three principles are readily modified to their Computer "C" versions:'

1C. Screen displays and menu systems should be organized functionally.
2c. Design the command language or menu display headings to be distinct from one another so as not to be easily confused, either in perception or in the action required.
3c. Make it difficult to do actions that can lead to operations with serious implications and that are not reversible.

LACK OF CONSISTENCY LEADS TO ERRORS

One class of errors occurs when a person attempts to rederive an action sequence and does so improperly, forming a sequence appropriate for an action different from the one intended. This occurs primarily through a lack of consistency in command structure. so that the appropriate structure for one command is not the same for another. even though the commands appear to be related and share a common description of purpose, action. and even part of the command format. Similar situations occur in the interpretation of instrument readings. The basic concept involved here is that when people lack knowledge about the proper operation of some aspect of a machine, they are apt to derive the operation by analogy with other. similar aspects of the device. The "derivation" may be unconscious, and it can influence behavior without the person realizing it. Forming conclusions from the relationships of one system to another is a common and powerful method of human thought, but it can lead to error if the mapping from one domain onto the other is not consistent (4, 2].

There are cases where lack of consistency seems desirable, and it is put into the design deliberately and with careful thought. This usually occurs when the normal sequence for an operation is long and tedious and when such an operation is to be performed frequently. Similarly, the default state of an instrument or control is sometimes made inconsistent with that of other instruments or controls because experience shows that the different defaults simplify some forms of operations. Nonetheless. these inconsistencies lead to errors and to difficulty in learning. One solution is to make command structure-and instrument formal consistent. even at the cost of some inefficiency of usage. A better solution would be to redesign the entire system to yield both consistency and ease of operation.

CAPTURE ERRORS IMPLY THE NEED TO AVOID OVERLAPPING COMMAND SEQUENCES

A *capture* error occurs when there is overlap in the sequence required for the performance of two different actions, especially when one is done considerably more frequently than the other. In the course of attempting the infrequent one, the more common act gets done instead. A capture error with the "vi" text editor on the Berkeley Release of the UNIX operating system occurs when attempting to write out a file. The command ":w" means to write the file, ":q" quits the editor (if the text has not been modified since the last writing of the file), and the combined sequence ":wq" writes. then quits. Because ":wq" is such a convenient operation. many people use it regularly as their way of finishing a day's session. and so it soon becomes an automatic command, with the status of a single operation rather than of two sequentially combined commands. However, as a result, at times when one wishes simply to write the file and continue with the editing. one finds oneself out of the editor and back in the operating system: by a capture error, the sequence ":wq" was typed instead of the simpler sequence ":w".

One possible way of avoiding this class of error is to minimize overlapping sequences, but this may not be possible. especially when the infrequent action sequence is simply a modification of the frequent one. In the case of "vi". if ":wq" were taken over by some other command (e.g., in newer versions of the system, "ZZ" is equivalent to ":wq"), the capture error would disappear, as the two different commands—":w" and "ZZ"—have no parts in common.

A second way of avoiding the error is to try to catch it where it occurs. The error occurs at the critical place where the sequences deviate. so it is here that the problem must be faced. If the system knows what the intention of the user is (perhaps by requiring the user to indicate the overall intention). it could be designed so that at the critical choice point the proper path was flagged or in some other way brought to the attention of the operator. In addition. sufficient feedback about the state of the system should be provided to provide reminders as to the deviation from the intention. A major issue here is simply to know the critical place at which the errors occur so that remedial action can be built into the system at that critical point.

'similar set of rules is given by Schneider. Wexelblat. and Jende [IS].

ACTIVATION ISSUES SUGGEST THE IMPORTANCE OF MEMORY REMINDERS

Activation errors are of two classes: inappropriate actions get performed and appropriate actions fail to get done. The former occurs when an inappropriate action sequence is activated either by being related to desired sequences (as in the capture error) or through events in the world (data-driven activation). The failure to carry out an action usually occurs because of memory failure; memory failure can occur when events intercede between the time of preparing an intention and the time the act should be performed. Various memory aids seem essential to prevent the latter. The first form of activation error may very well not be preventable. In this case, the system should be &signed to be tolerant of them.

In many ways the old saying, out **of *sight*, out** of ***mind***, is apt; if a set of operations is interrupted with other activities so that no reminder of them remains visible, the action **sequence is apt to be forgotten.** A good system **design will not let this happen, but will redisplay uncompleted sequences (or unanswered questions) whenever there is a chance that they are no** longer visible to the user. New high resolution displays with overlapping and reappearing windows handle this problem well (e.g.. the Xerox Star Computer System **[14]).**

PEOPLE WILL MAKE ERRORS, SO MAKE THE SYSTEM INSENSITIVE TO THEM

The analysis of errors provides one set of considerations for the **construction of a system that might minimize errors. There are several other factors that** should be considered as well. First, people will make errors even in the best designed systems, even with the best of training and motivations. So, a corollary of the attempt to minimize errors is that one should try to minimize the effect of an error. **This means that actions should be** reversible. at least as much as is possible. Some things, of course, once performed, are irrevocable. Actions that can lead to difficulty should be difficult to do, perhaps requiring a set of steps, as in the required release of "safeties" when the pilot wishes to eject from a military airplane, or at least, requiring a confirmation, as when requesting that all **files** on a computer directory be destroyed. Schneider, **Wexel**blat. and **Jende [13] suggest that irreversible actions should require considerable "mental force"** in order to be performed, a useful way of viewing the need to make these operations difficult to do.

It is not sufficient to ask the user to confirm that a **particular** action sequence is wanted, because if confirmation is routinely asked for (and if the usual response is 'yes"). the confirmation itself becomes an automatically invoked component of the command sequence. Thus, if the command is given in error, it is likely to have the confirmation invoked as part of the same error; in our experience, the confirmation is as apt to be in error as much as the original command. As Newman points out in his discussion of the paper by Schneider et al.. the normal response to requests for confirmation is something like this: "Yes. yes, yes, yes. Oh dear!" The point is that disastrous commands should be difficult to carry out: confirmations of the validity of the command may not offer **sufficient** difficulty to be a satisfactory safeguard.

Sometimes the command can act as if it were actually executed, when in fact, it has only been deferred. **Con**aider the command to delete files from the system; the system could claim that it has removed the file, but has **actually** put it away on some temporary location so that it can be recovered later if its **"deletion" was discovered to have been an error.** (Real **deletion can be** done on an infrequent basis, say after a lapse of several hours or days.) In Interlisp **[15] operations may he** "undone," even operations such as writing on or destroying **files.**

LESSONS

These simple observations lead us to some conclusions about system design. Obviously, this analysis only takes us part of the way toward a set of design principles. An analysis of errors can only get at some classes of problems, and there may not always be general rules applicable to the issues. These analyses must be supplemented with other methods, including an understanding of the nature *of* **a person's** mental image of the system and of **the** human information processing capability of the user. Meanwhile. the analyses presented here make several points:

Feedback: The state of the system should be clearly available to the user. ideally in a form that is unambiguous and that makes the set of options readily available so as to avoid mode errors.

Similarity of response ***sequences:*** Different classes of actions should have quite dissimilar command sequences (or menu patterns) so as to avoid capture and description errors.'

Actions should ***be*** reversible (as much as possible) and where both irreversible and of relatively high consequence, they should be difficult to do, thereby preventing unintentional performance.

Consistency of the system: **The** system should be consistent in its structure and design of command so as to minimize memory problems in retrieving the operations.

The desired goal is the establishment of a discipline of Cognitive Engineering" that can provide designers with the tools required to make their products more sensitive and responsive to the needs of the users. These tools will have at least two components: first, a set of well-established procedures and methods with known benefits and costs, advantages, and disadvantages; second, a set of quantitative modeling aids that can be used to assess the performance to be expected from a particular design choice. My hope is that by providing a rationale based upon modern cognitive theory, it will be possible to generalize these findings to new situations and to present them in such a way that designers will find them accessible and **useable** during the course of design. There are three methods that provide useful beginnings:

1. **Start with** the psychological mechanisms; use knowledge of the processing mechanisms to derive the important constraints on human performance. This is the approach taken by Card, Moran, and Newell **[1].**
2. Base the design around the mental models of the user. People form mental models of each other, the world, and the devices and systems with which they interact. These mental models are used to predict system behavior and guide actions. People's mental models, however, have interesting properties, sometimes being derived from idiosyncratic interpretations of the system. Moreover, their models must operate within the constraints of the human processing system. The

study of these models, their veracity, and the capability of people to use the models wisely provides an important tool for the understanding of the human-system interface and for the development of design rules. This is the approach described in the book edited by Gentner and Stevens, *Mental* Models [3] and in Norman [10, 11).

3. Use analyses of people's performance in a variety of situations-but especially their errors-to construct an analysis of the appropriate form of human-machine interface that would optimize performance and minimize either the incidence of error or the effect of the error. This is the theme of this paper.

All three of these approaches are complementary and should be combined in any complete attempt to produce a cognitive engineering approach. This brief paper has demonstrated the principles underlying the third approach.

REFERENCES

1. Card, S.. Moran. T., and Newell. A. Applied *Information-Processing* Psychology: The Human-Computer *Interface*. Erlbaum Associates. Hillsdale, N.J.. 1983.
2. Gentner, D. The Structure *of Analogical* Models in Science: *Technical Report 4451*. Bolt. Beranek and Newman, Cambridge, Mass. 1980.
3. Gentner, D. and Stevens. A. (Eds.), *Mental* Models. Erlbaum Associates, Hillsdale, N.J., 1983.
4. Lakoff, G. and Johnson. M. Metaphors We Live By. University of Chicago press. Chicago, 1980.
5. Lockheed Missiles and Space Company. Human factors review of nuclear power plant control room design. EPRI NP-309-SY. Electric Power Research Institute. Palo Alto. California. 1976.
6. National Transportation Safety Board (NTSB). *Aircraft* incident report: *Aero* Mexico *DC-10-30, XA-DUH* over *Luxembourg*, Europe. November It. 1979. (NTSB-AAR-80-10). National Transportation Safely Board. Washington. D.C., November. 1980. (Report available from National Technical Information Service.)
7. National Transportation Safety Board (NTSB). *Special investigation* report: Design-induced landing gear retraction accidents in Beech Boron. *Bonanza, and other light aircraft*. (NTSB-SR-80-1). National Transportation Safety Board. Washington, DC.. June. 1980. (Report available fmm National Technical Information Service.)
8. Norman, D. A. Categorization of action slips. Psychological Review. 88, 1981, I-15.
9. Norman. D. A. A psychologist views human processing: Human errors and other phenomena suggest processing mechanisms. Invited address, published in Proceedings of *the International* joint Conference on *Artificial Intelligence*. Vancouver, 1981.
10. Norman, D. A. How to make computer systems that people will like to use: Steps toward a cognitive engineering. Proceedings *of the Western Electronic Show and* Convention *(Wescon/82)*. Anaheim, California. Sept. 15, 1982.
11. Norman. D. A. Some observations on mental models. In D. Gentner and A. Stevens (Eds.), *Mental* Models. Erlbaum Associates, Hillsdale, N.J., 1983.
12. Norman, D. A. and Bobrow, D. G. Descriptions: An intermediate stage in memory retrieval. *Cognitive* Psychology, *11*, 1979, 107-123.
13. Schneider. M. L. Wexelblat, R. L., and Jende, M. S. Designing control languages fmm the user's perspective. In D. Beech (Ed.). Commond *Language* Directions: Proceedings of the *IFIP TC* 2.7 Working Conference on Command *Languages*. North-Holland Publishing Co., New York. 1980.
14. Smith, D. C.. Irby, C.. Kimball. R.. and Verplank, B. Designing the star user interface. Byte 7. 4 (April 1982), 242-282.
15. Teitelman, W. and Masinter, L. The Interlisp programming environmeat. Computer *14*. 4 (April 1981). 25-33.

CR Categories and Subject Descriptors: H.1.2 [User/Machine Systems] Human Information Processing
General Terms: Human factors
Additional Kay Words and Phrases: human errors, software design

Received 3/82; revised ond accepted 12/82