Int. J. Man-Machine Studies (1984), **21**, 365-375

Stages and levels in human–machine interaction

DONALD A. NORMAN
Department of Psychology and Institute for Cognitive Science C-015, University of California, San Diego, La Jolla, California 92093, U.S.A.

The interaction between a person and a computer system involves four different stages of activities—*intention*, *selection*, *execution*, and *evaluation*—each of which may occur at different levels of specification. Analysis of these stages and levels provides a useful way of looking at the issues of human–computer interaction.

Introduction

My concern is with the overall process of interaction with the computer. I want to avoid an emphasis on detailed aspects of that interaction and ask about the nature of the interaction. Details are indeed important, but only once the proper conceptualization has been applied. Consider a simple situation. A user of a computer system is writing a paper and, in the process, decides that the appearance of the printed draft is not ideal: the paragraph indentation does not look proper. The user forms an intention: to correct the appearance of the paper. Now the problem is to satisfy this intention by translating it into the appropriate set of actions. The purpose of this article is to examine some aspects of the interaction between a person and the computer system as the person attempts to satisfy the intention. The focus is derived from three observations.

1. When a person interacts with a computer, it is possible to identify four different stages of that interaction, each with different goals, different methods, and different needs (Norman, 1984).
2. Each of the known techniques for the interface has different virtues and different deficiencies. Any given method appears to lead to a series of trade-offs. Moreover, the trade-offs differ across the four stages of user interaction (Norman, 1983*a*).
3. Messages and interactions between user and machine can take place at a number of different levels. If the levels are not matched, confusion and misunderstanding can arise. Determining the appropriate level is a difficult task, often requiring some knowledge of the intentions of the user (Norman, 1981*a*, 1983*b*).

Let us start with a brief analysis of the stages.

The four stages of user activities

I define *intention* as the internal, mental characterization of the desired goal. Intention is the internal specification of action responsible for the initiation and guidance of the resulting activity. Although intentions are often conscious, they need not be. *Selection*

0020-7373/84/100365+11$03.00/0

is the stage of translating the intention into one of the actions possible at the moment. To go from intention to action, the person must review the available operations and select those that seem most auspicious for the satisfaction of the intention. Then, having mentally selected, the actual command sequences must be specified to the computer. The determination of a particular command or command sequence is *selection*; the act of entering the selections into the system is *execution. Intention* and *selection* are mental activities; *execution* involves the physical act of entering information into the computer. These activities do not complete the task. The results of the actions need *evaluation*, and that evaluation is used to direct further activity.

Thus, the full cycle of stages for a given interaction involves:

forming the intention;
selecting an action;
executing the action; and
evaluating the outcome.

Perhaps the best way to understand the differences among the stages is to continue with our example. The intention is to improve the appearance of the printed version of the manuscript. This is a higher order statement that must get translated into more specific terms. Suppose that because it is the paragraph indentation that looks wrong, the user decides to switch to a "block paragraph" format—a format in which the initial line of a paragraph is not indented. We now have a second level of intention: call the main intention *intention*$_0$ and this new level *intention*$_1$. But even *intention*$_1$ is not sufficiently specific. Suppose the manuscript is being prepared by means of a traditional editor and run-off facility, so the manuscript contains formatting instructions that get interpreted at run-off time. One way to carry out the intention is to change the definition of the paragraph. Another way is to bypass the paragraph format specification and to substitute a blank line instead. There are several ways of carrying out each of these methods, but suppose that our user decides upon the latter technique, substituting for the paragraph specification *.pp*, the "skip a line" specification, *.sp*. This becomes *intention*$_2$.

Having formed *intention*$_2$, the next step is to *select* an appropriate set of actions to carry it out. This requires a set of text-editing commands, commands that find the appropriate location in the text that is to be changed (in this example, there are apt to be a rather large number of locations), then commands that change the *.pp* to *.sp*. There are several different ways of performing these operations. Thus, in the particular text editor that I happen to be using to write this paper (Berkeley UNIX *vi*), the following command sequence will do the operation *g/ ^.pp/s/pp/sp/*.† A more detailed analysis of the steps involved in making the selection would reveal that several more levels of intentions were involved. Eventually, however, a set of text-editing commands will be selected. We must take note of one more level of intention: the intention to execute the selected command sequence. Call this *intention*$_3$.

† As with many text editors, the command sequence is not particularly intelligible. The initial *g* signals that the command is to be performed "globally" to all occurrences of the string. The */^.pp/* is the string that is searched for in the text: a line that begins with "*.pp*" The remaining part of the line specifies the *substitute* command: substitute for the string "pp" the string "sp". Users of the *vi* text editor will recognize that even this description is slightly simplified. It should be clear that selecting this command string is a reasonably complex operation, requiring the setting of numerous sub-intentions and engaging in some problem-solving.

Having selected the command sequence, the next step is to *execute* the selection. In *vi*, this will require yet another action cycle to get the editor into the mode in which the substitute command will work properly, an action cycle that requires yet more levels of intentions, selection, execution, and evaluation. Finally, if all has gone well, the user has executed $intention_3$, and entered the command sequence into the system. Although *execution* has its own cycle of activities and sub-intentions, let us skip over them and assume that this stage has been performed properly.

This brings us to *evaluation.* Evaluation has to occur separately for each level of intention. First, it is necessary to check that the command sequence entered into the editor is the one intended. Then the manuscript text must be examined to make sure that $intention_3$ (the global change) got properly carried out: that all the *.pp* lines do indeed now say *.sp.* If they do, $intention_2$ (change pp to sp) has also been satisfied. Then $intention_1$ (change to block paragraph format) must be evaluated by means of yet another action cycle and another set of intentions. To see if the paragraphs come out in desired block-paragraph style, it is necessary to "run-off" the manuscript: this involves a new intention, $intention_{1A}$, and a new selection of commands. When all that is complete, the user can finally examine the printed page and determine whether $intention_1$ has been satisfied. If so, then the outcome can be evaluated with respect to $intention_0$ to determine whether the new format is a satisfactory improvement over the original.

STAGES ARE APPROXIMATIONS

Note that although it is useful to identify stages of user activity, the stages should be thought of as convenient approximations, not as well-defined, well-demarcated psychological states. People are not serial-processing mechanisms, they do not have well-defined stages of decision processes or action formation, and they often are not conscious of the reasons for their own actions. People are best viewed as highly-parallel processors with both conscious and subconscious processing, and with multiple factors continually interacting and competing to shape activity (see Norman, 1981*a*, *b*; Rumelhart & Norman, 1982). Nevertheless, the approximations used by this analysis may yield relevant and worthwhile results for the identification of important design considerations.

THE INTENTION STAGE

From the point of view of a system designer, there are two different aspects of intentions, each of which can be divided into two different concerns. The first aspect is the system's need (and ability) to know the intentions of the user, the second is the support that can be offered to the user to help form appropriate sub-intentions.

Knowing the user's intentions

Consider what the system might need to know about the user's intentions. There are two concerns here: (a) *what* needs to be known about a user's intentions, and (b) *how* it is possible for a computer system to get this information. The problem is made more complex because of the multiple-layers of intentions that exist, with any reasonable task involving a fairly complex structure of intentions and sub-intentions. Still, for a system to provide useful guidance and feedback, it is going to need information about

the user's higher-level intentions, both the overall, general intention and the sub-intention that is relevant at the moment (and perhaps the entire chain from the current sub-intention back to the highest level intention). Indeed, one could argue that all assistance (including help and error messages) requires input about the higher levels of user intentions in order to be maximally effective (see Johnson, Draper & Soloway, 1983). The second concern, *how* a system can get the necessary information about the user's intentions, is the difficult one. In some cases, the user can simply be asked. In others, it will be far more complicated. I expect that as we learn more about *what* the higher-levels of intention relevant to the task are, we will go a long way toward solving the *how* problem.

System support for sub-intention formation

There are usually two things a user needs to known in forming intentions: what the current status of things is and what is possible, given the current status and system facilities (both of these points are also appropriate for other stages: the question "what is the current status?" is part of evaluation; the question "what is possible?" is part of selection). We need to learn how to provide this information, at the appropriate level of sophistication for a given user at a given task, without intrusion.

THE SELECTION STAGE

Some intentions might map directly onto a single action, others might require a sequence of operations. In either case, the selection of an action sequence can require considerable knowledge on the part of the users. There are two aspects of selection. One is to figure out the method that is to be used in doing the task, the other to select which particular system commands are to be invoked. Consider how users decide what the options are in the selection process. How do they know the commands? There are four ways.

1. They could retrieve them from memory.
2. They might be reminded, either by another person, the system, or a manual.
3. They might be able to construct or derive the possibilities.
4. They might have to be taught, either by another person, the system, or a manual.

In the first case, recall-memory is used to identify the desired item. In the second case, recognition-memory is used to identify the desired item from the list or description of the alternatives. In the third case, the user engages in problem-solving, perhaps using analogy, perhaps eliminating possibilities. And in the fourth case, the user learns from some external source. This raises the issue of how the user knew that assistance was needed and how that assistance was then provided—a major theme of study in its own right.

Support for the selection stage comes principally from memory aids (manuals and various on-line support tools such as menus, help commands and icons) that allow the user to determine the possible commands and their modes of operation, prerequisities and implications. Selection can be enhanced by "workbenches" that collect together relevant files and software support in one convenient location. Other methods of structuring groups of commands and files dependent upon the user's intentions need to be explored (for example, see Bannon, Cypher, Greenspan, & Monty, 1983).

THE EXECUTION STAGE

Naming

There are two ways to specify an action to the computer: *naming* and *pointing*. Naming is the standard situation for most computer systems. The designer provides a command language and the users specify the desired action by naming it, usually typing the appropriate command language sequences. A speech input system would also be executing by naming. Execution by naming provides the designer with a number of issues to worry about. What is the form of the command language? How are the commands to be named, how are options to be specified? How are ill-formed sequences to be handled? How much support should be provided for the user?

Most operating systems provide little or no support for intention, selection or execution. The user is expected to have learned the appropriate commands. Then the execution is judged either to be legitimate (and therefore carried out) or erroneous (and an error message presented to the user).

It is quite possible for a system to provide considerable support for these stages, to provide information that tells not only the actions available, but also the exact procedure for executing them. This can be done with menus, perhaps abbreviated and restricted in content, so that they serve as reminders of the major actions available.

Pointing

Execution of an action by pointing means that the alternative actions are visually present and that the user physically moves some pointing device to indicate which of the displayed actions are to be performed. Although the prototypical "pointing" operation is to touch the desired alternative with a finger or other pointing device, the definition can be generalized to include any situation where a selection is made by moving an indicator to the desired location.

Note that a naming system requires two things: a place to point at and a means of pointing. We can separate these two. Moreover, as long as one needs a place to point at, it might as well be informative. Thus, the places serve as reminders to the selection stage when they consist of printed labels, lists, menus, or suggestive icons displayed on a terminal screen. But the places need not be informative: they might be unlabelled locations on the screen (or, in electronic devices, unlabelled—or illegible—panels).

Executing by naming often allows a large set of possible actions, hard to learn, but efficient in operation. Execution by pointing is restricted to those commands that can have a specified location. As a result, proponents of naming systems say they are more efficient: pointing requires sublevels. Proponents of pointing say they are easier to remember. One side emphasizes ease of execution, the other ease of selection.

THE EVALUATION STAGE

Feedback is an integral part of evaluation, whether the operation has been completed successfully or whether it has failed. For full analysis, the user must know a number of things.

What the previous state of the system was.
What the intentions were.
What action was executed.
What happened.

How the results correspond to the intentions and expectations.
What alternatives are now possible.

The evaluation of an action depends upon the user's intentions for that action. In cases where the operation could not be performed, either because it was not executed properly, or because some necessary precondition was not satisfied, the user will probably maintain the same intention but attempt to correct whatever was inappropriate and then repeat the attempt. In cases where the operation was done, but with undesirable results, the user may need to "undo" the operation. In this case, repetition of the same action is not wanted.

One useful viewpoint is to think of all actions as iterations toward a goal. Ill-formed commands are to be thought of as partial descriptions of the proper command: they are approximations. This means that error messages and other forms of feedback must be sensitive to the intentions of the users, and, wherever possible, provide assistance that allows for modification of the execution and convergence upon the proper set of actions.

The user support relevant to each stage is summarized in Table 1.

TABLE 1
Design implications for the stages of user activities

Stage	Tools to consider
Forming the intention	Structured activities Workbenches Memory aids Menus Explicit statement of intentions
Selecting the action	Memory aids Menus
Executing the action	Ease of specification Memory aids Menus Naming (command languages) Pointing
Evaluating the outcome	Sufficient workspace Information required depends on intentions Actions are iterations toward goal Errors as partial descriptions Ease of correction Messages should depend upon intentions

Interface aids

MENUS IN THE FOUR STAGES

One of the more common interface aids is a menu, implemented either as a set of verbal statements or as pictures ("icons"). It is useful to examine menus at this point

for two reasons. First, the use of menus is often controversial, in part because their use requires trading the perceived value of the information provided by the menu for a loss of workspace and a time penalty [these trade-offs are discussed in Norman (1983a)]. Second, two different aspects of menus are often confounded. Menus can serve as a source of information for the intention and selections stages. In addition, they can also provide information, or even the mechanism, for the execution stage. That is, in execution by pointing, the menu or icon provides both information and a place to point. Unnecessary confusion arises when these rules of menus for selection are lumped with their roles for execution. Menus as sources of information for the intention and selection stages have one set of virtues and deficits; menus as mechanisms for the execution stage have another set of virtues and deficits. The point is that menus serve different purposes and have different trade-off values for each stage: in part, the virtues for one stage are pitted against the deficits for another. The properties of menus can be summarized in the following way.

I. Menus are capable of providing information for *intention* and *selection* by:
   A. Presenting the user with a list of the alternatives;
   B. Presenting descriptions and explanations of the alternatives.
II. Menus can aid in the *execution* stage:
   A. if execution is by *pointing*, menus can aid by:
      1. Providing a target to be pointed at.
   B. If execution is by *naming*, menus can aid by:
      1. Providing the user with an abbreviated execution name (such as the number of the menu line, a single letter, or a short abbreviation, usually mnemonic);
      2. Providing the user with the full command line (and arguments) that are to be used.

The first function of menus, providing information, is really their primary function. The information, explanations and descriptions that they present are especially important in the stages of forming the intention and selecting the action. Note that this function can be performed without any commitment to how execution is carried out. The second function of menus, aiding in execution, can be of equal use for execution by naming or pointing. Menus are especially useful when only a restricted number of alternatives is available, usually restricted to those described by the menu. Execution might be performed either by pointing at the menu items or by typing simplified command names (which are often so configured as to only require the typing of single characters).

Another major design decision is the question of how to get access to menus. The alternatives for menus are the following.

1. Always to be present in full form. Note that a set of labelled function keys can be thought of as a menu that is always present, with execution by pointing (i.e. depressing the appropriate key). In this sense, then, the panels of conventional instruments use a form of menus; the set of controls and range of possible actions are always visible. This option optimizes access to information at the expense of workspace.

2. Always to be present in a reduced form that allows the user to request the full menu. This option is a compromise position between the demands for information and workspace.

3. Not to be present unless requested by a special command or labelled key (e.g. "help") or by some other action (e.g. a "pop-up" menu called by depressing a button on a mouse). This option maximizes workspace at the expense of time and effort.

4. Available through a hierarchical or network structure, necessary when the menu size is large.

Note that fans of menus usually are those who weight highly the information provided for intention and selection. Foes of menus usually are those who do not need assistance in these stages and who object to the loss of time and workspace during execution and evaluation. The differences come from differing needs at the different stages. Table 2 summarizes the effects of these issues on menu design.

TABLE 2
Properties of Menus

Variable	Virtues	Deficits
Information	The more information presented in one display, the more detailed the explanations can be or the more alternatives can be presented, in either case improving the quality of advice offered the user	More information increases times for searching, reading, and displaying, making it harder to find any given item, decreasing usability and user satisfaction
Amount of workspace used	The more workspace available for the menu, the more information can be displayed and the better it can be formatted, simplifying search and improving intelligibility	The larger the percentage of the available space used, the more interference with other uses of that space
Display of a large number of menu items	Allows user to see a large percentage of the alternatives, aiding intention and selection stages and minimizing number of menus needed	Slow to read, slow to display, uses large percentage of the available space
Display of a small number of menu items	Easy to read, quick to display, only a small percentage of the available space is required	If number of alternatives is large, multiple menus must be provided. This can be slow and cumbersome

Levels of activity

THE PROBLEM OF LEVELS

The existence of numerous levels of intentions leads to numerous difficulties. First, there can be a mismatch between the level at which the user wishes to express the intentions and the level that the system requires. Second, even apparently simple tasks can require considerable levels of intentions and sub-intentions, and the person's short-term memory may become overloaded, leading to confusion and error.† Finally,

† A number of "slips" of action occur for this reason, where the person loses track of the higher-order intention but continues to perform the actions associated with the lower-order ones. The result is to perform some action, only to wonder why the action is being done. When the lower-level actions are completed, there might no longer be any trace of the originating intention/action [an example from my collection: walk across the house to the kitchen, open the refrigerator door, then say "Why am I here?" (Norman, 1981*a*)].

there can be difficulties in the evaluation stage, especially when the results are not as expected. Here the problem is to determine at what level the mismatch occurs. An example from a program on our system illustrates the problem. I wish to display the contents of a file on the screen. I execute the appropriate display program and it works well. However, when I try to perform one of the options of the display program, the program collapses most ungracefully, and then displays this message on the screen: *longjmp botch: core dumped.* What is a "longjmp botch"? Why am I being told this? Of what use is this information to me?

The message was obviously written by a conscientious programmer who perhaps thought the situation would never arise, but that when it did, it would be important to tell the user. † One problem with this message is that it is presented at the lowest level of program execution whereas I am thinking at a fairly high level of intention: I want to change what material is on the screen and want it either to be done or to see a message telling me that it cannot be done, in reasons relevant to my level of thought. "longjmp botch" is not the level at which I am forming my intentions.

Remember the earlier example of attempting to reformat the paper. Suppose the end result is not satisfactory. Why not? The reason could lie at any level. Perhaps the run-off was not carried out properly; perhaps the change of *.pp* to *.sp* was not done properly; perhaps that change did not properly perform the "block paragraph" formatting; perhaps "block paragraph" is not what is required to satisfy $intention_0$. There are many places for error, many places where intentions could fail to be satisfied. If the operation were carried out manually, one step at a time, then it would be relatively easy to detect the place where the problem lies. But in many situations this is not possible: all we know is that the intention has not been satisfied. Many of us have experienced this problem, spending many hours "fixing" the wrong part of a program or task because we did not have the information required to judge the level at which the problem had occurred. The question, however, for the system designer is: what information is most useful for the user?

The question is very difficult to answer. For the system programmer who is trying to debug the basic routines, the statement *longjmp botch* might be very useful—just the information that was needed. For me, it was worthless and frustrating. A statement like *System difficulties: forced to abort the display command* might have been much more effective for my purposes, but rather useless to the systems programmer. The problem is not that the error message is inappropriate; the problem is that sometimes it is appropriate, other times not.

One solution to the levels problem is to know the intention. If the program knew it was being used by a person who only wanted to see the files, it could make one set of responses. If it knew it was being used by someone trying to track down a problem, it could make another set. However, although knowing user intentions and levels often helps, it does not guarantee success. In my studies of human errors I have found numerous cases where knowledge of the intention would not help. Consider the

† It is from this and related experiences that I formulated the rule: programmers should never be allowed to communicate with the user. Good software design, I am convinced, can only come about when the part of the program that communicates with the user is encapsulated as a separate module of the program, written and maintained by an interface designer. Other parts of the program can communicate only with each other and with the interface module—most definitely *not* with the user. See Draper & Norman (1984).

following example:

> X leaves work and goes to his car in the parking lot. X inserts the key in the door, but the door will not open. X tries the key a second time: it still doesn't work. Puzzled, X reverses the key, then examines all the keys on the key ring to see if the correct key is being used. X then tries once more, walks around to the other door of the car to try yet again. In walking around, X notes that this is the incorrect car. X then goes to his own car and unlocks the door without difficulty.

I have a collection of examples like this, some involving cars, others involving apartments, offices, and homes. The common theme is that even though people may know their own intentions, they seem to tackle the problem at the lowest level, and then slowly, almost reluctantly, they pop up to higher levels of action and intention. If the door will not unlock, perhaps the key is not inserted properly. If it still will not work, perhaps it is the wrong key, and then, perhaps the door or the lock is stuck or broken. Determining that the attempt is being made at the wrong door seems difficult. Now perhaps the problem is the error messages are inappropriate: the door simply refuses to open. It would be better if the door could examine the key and respond "This key is for a different car". Can programs overcome this problem?

This article is intended only to introduce the ideas that there are stages of activity, levels of intention and trade-offs among the solutions to the problems of human-user interaction. As the saying goes, more work is needed. But if that message is understood, then the article is successful. My goal is to move the level of study of the human interface up, away from concentration upon the details of the interaction to consideration of the global issues.

The ideas discussed here result from the interactions with the UCSD Human–Machine Interaction project. The various sections of the paper have been presented at the SIGCHI Conference on Computer–Human Interaction (Norman, 1983*a*), the IFIPS First Conference on Human–Computer Interaction (Norman, 1984), and at the NSF Conference on Intelligent Interfaces, New Hampshire, 1983. Sondra Buffett and Edwina Rissland have provided helpful critiques of various drafts of the article.

The research was supported by Contract N00014–79–C–0323, NR 667–437 with the Personnel and Training Research Programs of the Office of Naval Research and by a grant from the System Development Foundation. Requests for reprints should be sent to Donald A. Norman, Institute for Cognitive Science C–015; University of California, San Diego; La Jolla, California 92093, U.S.A.

References

BANNON, L., CYPHER, A., GREENSPAN, S. & MONTY, M. L. (1983). Evaluation and analysis of users' activity organization. *Proceedings of the CHI 1983 Conference on Human Factors in Computer Systems*, Boston.

DRAPER, S. & NORMAN, D. (1984). Software engineering for user interfaces. *Proceedings of the 7th International Conference on Software Engineering*, Orlando, Florida.

JOHNSON, W. L., DRAPER, S. & SOLOWAY, E. (1983). Classifying bugs is a tricky business. *ACM SIGSOFT/SIGPLAN Symposium on High-Level Debugging*, Baltimore, Maryland.

NORMAN, D. A. (1981*a*). Categorization of action slips. *Psychological Review*, **88**, 1–15.

NORMAN, D. A. (1981*b*) A psychologist views human processing: Human errors and other phenomena suggest processing mechanisms. *Proceedings of the International Joint Conference on Artificial Intelligence*, Vancouver.

NORMAN, D. A. (1983*a*). Design principles for human–computer interfaces. *Proceedings of the CHI 1983 Conference on Human Factors in Computing Systems*, Boston.

NORMAN, D. A. (1983*b*). On human error: Misdiagnosis and failure to detect the misdiagnosis. Talk presented at the *GA Technologies Inc. Workshop on Decision Processes in Operation Planning and Fault Diagnosis*, La Jolla, California.

NORMAN, D. A. (1984). Four stages of user activities. In SHACKEL, B., Ed., *INTERACT '84, First Conference on Human-Computer Interaction*, Amsterdam: North Holland.

RUMELHART, D. E. & NORMAN, D. A. (1982). Simulating a skilled typist: A study of cognitive-motor performance. *Cognitive Science*, **6**, 1–36.