ISSN 0105-8517

Learning Difficulties of Programmers Faced With a New Programming Environment

Jakob Nielsen

DAIMI PB - 181
October 1984

Computer Science Department
AARHUS UNIVERSITY
Ny Munkegade – DK 8000 Aarhus C – DENMARK
Telephone: 06 – 12 83 55

PB - 181 **J. Nielsen: Learning Difficulties ...**

TRYK: DAIMI/RECAU

Learning Difficulties of Programmers Faced With a New Programming Environment

Jakob Nielsen
Aarhus University
Denmark

Abstract

When faced with a totally new environment composed of a new interaction paradigm and a new programming tool, programmers exhibited many of the same learning difficulties as novice users learning to use an ordinary word processor.

Keywords and phrases:

Learning Difficulties, Programmers versus Novices, Structure Editing, Interaction Paradigms.

1. Introduction

When discussing software ergonomics it is common to make the distinction between programmers on the one hand and non-programmers on the other hand. Programmers are usually assumed to be able to quickly learn how to use a new computer system and to have less trouble than other users in general.

Many studies on learning difficulties have focused on novice users without programming experience. In this study we have investigated the learning problems of programmers when they are placed in a situation analogous to that faced by naive users. Our learners had to use a system with a functionality that was unfamiliar to them and at the same time they had to use a computer system based on a totally different interaction scheme than the one they were used to. These two aspects of the new environment are described in sections 2 and 3 respectively.

2. Structure oriented editing and the EKKO-editor

The experiment described in this paper was carried out with the EKKO-editor (Borup et al. 1983, Nørmark 1985) which is a structure oriented editor. The present version of EKKO is an editor for Pascal. Structure editors for specific programming languages are often called syntax directed editors.

A structure oriented editor knows the syntax of the programming language it is used for and it can only generate syntactically correct programs.

Several other syntax directed editors exist but they have usually not been the object of formal software ergonomics experimentation. Since the intended users of such an editor are programmers and since the editor designers are themselves programmers, the designers may feel that they are justified in doing the human factors evaluation of their editor using themselves as subjects, see e.g. (Zelkowitz 1984).

The editor developers and the editor users are two different kinds of programmers however. In the course of designing and implementing their editor, the developers form a detailed actual conceptual model of the editing principles implied by a structure editor as opposed to a text editor. The users do not have this model, and the present experiment showed that ordinary programmers may indeed have several problems when faced with a totally new way of editing.

The basic principle of structure editing is that the editing process follows the underlying structure of the object being edited (e.g. the syntax of a program) and not the textual surface view of the object. This principle is sufficently new to present learning problems to the programmers. Usually they only have to learn a new command language of some computer system working in essentially the same manner as they are used to. But in this experiment they have to learn both a new command language and a new basic functionality at the same time.

3. The paradigms of interaction

EKKO runs on a PERQ graphical workstation and uses the modern interaction paradigm with windows as output medium and the combination of a mouse and a keyboard as input media.

We may according to (Card et al. 1984) classify computer displays according to which one out of several described types of windowing is used. We could actually make an even more fundamental division into several paradigms of interaction, even though these paradigms turn out to be connected to the windowing mechanism used.

We will describe four interaction paradigms. Each paradigm is the foundation for lots of different computer systems, but it is usually quite easy to classify a given system as based on some specific paradigm. Closely coupled with each paradigm is a set of basic rules for how to communicate with the system, and once a user has learnt the rules for some paradigm, they form the framework for rapidly learning how to use other systems based on the same paradigm.

When a user has to learn a system based on some other interaction paradigm, the user cannot rely on the existing framework but has to build another. Therefore some additional learning difficulties may be expected until the user has acquired a working actual conceptual model of the new interaction paradigm and can use this model as a basis for learning the systems implemented in this paradigm.

The four existing interaction paradigms are:

1) No interaction! This corresponds to batch systems.

2) TTY. This is one-line-at-a-time systems. They usually employ traditional command languages based on keywords etc.

3) *Full screen*. The whole of the display screen forms the unit of interaction. Screen layout is an important discipline for designers in this paradigm. Often a very hierarchical menu structure is employed for controlling the interaction.
4) *Window systems*. The display is divided into several windows so that different kinds of information is displayed in different windows. Control if often via pop-up menus and the interaction style is as mode-less as possible. A mouse or other non-keyboard device is used for pointing which is the selection method instead of naming. Icons and other non-animated graphics are used.

It is interesting to consider how the fifth interaction paradigm will look. It may include voice I/O and animated pictures instead of still graphics. But this is not the topic of the present paper.

4. Experimental method

Three computer science students used the EKKO editor for five days each and two professional programmers from industry used the editor for two days each. All 5 subjects were men. None of the test subjects had used either a structure oriented editor or the PERQ workstation before. They had had experience with several different kinds of text editors, both line oriented and full screen editors.

The small number of subjects precluded the use of statistical measurements but made possible a more detailed analysis using such methods as thinking aloud and interviews. Also we wanted to have each test subject use the editor for so long that he became comfortable using it so that we could study both learning behavior and stady state performance. All input made by the test subjects was recorded on a file with a timestamp, making a complete replay of a session possible so that problems could be studied in detail (Neal et al. 1984). Since mouse movement is a continuous process, it would have required quite a lot of data to record the total movement accurately. Instead we chose to record the position of the cursor on the display only when some mouse operation was carried out.

Each test subject first read the editor manual (for ca. 1 hour) and then for ca. 2 hours followed a self paced practice assignment that covered all important aspects of the editor. After this the three student subjects all carried out the same experiments consisting of a mixture of three types of tasks:
1) The creation of program text from common existing paper manuscripts using the editor.
2) Online programming to solve common specific problems.
3) Making specified changes to their own programs, generated by one of the other two types of experiment.

An example of a type 1 experiment was to reproduce a program that counted the number of lines in a file. An example of a type 2 experiment was to program a simple payroll application. A type 3 experiment was then to change the payroll program to include days of absence due to employee sickness. The programs were of a size of between 20 and 200 lines of Pascal code.

The professional programmers also carried out a mixture of the three types of tasks but only had time for a subset of the experiments carried out by the students.

5. Learning difficulties

Our subjects had to learn the editor by unaided self-study. We found that they had almost the same problems as those found by Mack et al. (1983) even though they investigated computer naive typists learning a simple text based word processor. The main difference was that our learners had the courage and prerequisites for charging ahead whenever they got stuck. They simply generated sufficiently many alternative possible solution procedures for each problem until one of them worked.

We will report some of the observed learning difficulties using the classification of 8 headings from (Mack et al. 1983) to structure the discussion. One further category is presented, namely "Over-estimating the computer".

The 9 categories of learning difficulties are not disjoint. Several problems belong to more than one category, but in the discussion to follow we will place concrete examples of learning problems in the category where they seem most fitting.

5.1. Learning is difficult

Mack et al. report that their learners became frustrated and complained about the amount of material to read. They took quite a long time to learn. Our participants were not able to use the editor "fluently" the first day and they did have problems acquiring the relatively large number of new concepts in both structure editing and window-based user interfaces at the same time. On the other hand our learners did not experience nearly the same frustrations as the learners in the Mack et al. experiment, since our learners had experience with the acquisition of many different computer concepts. Frustrations are a function of both the actual trouble experienced by people and of their expectations. Our subjects had already learnt to expect some problems when faced with a new computer system, so they did not become frustrated.

5.2. Learners lack basic knowledge

When faced with a new interaction paradigm, learners lack general knowledge of the way the interface works and they fail to understand basic ideas and jargon. Since any interaction paradigm is new to computer-naive learners, they could be expected to experience especially serious problems in this category as shown in (Mack et al. 1983).

Our users were certainly not computer naive, but several problems were nevertheless noted in this category. Many problems were due to the use of an interaction paradigm using windows and mouse-controlled menus. For example, all our users had problems the first time they created a long line in a program. The line would not fit inside the window and part of the line was therefore invisible. It took some time for our learners to realize that the missing part of the line had not been dropped by the editor but was still present inside the computer and could be worked on. They had trouble thinking of the concept of horizontal scrolling, which of course may also be due to their considering a program an one-dimensional structure that can only be scrolled vertically.

5.3. Learners make ad hoc interpretations

Users try to make sense of what they experience by constructing and elaborating ad hoc interpretations of their experiences. An example: Instead of typing in an identifier you may in EKKO copy it from an existing occurence. The intended conceptual model of this operation is that "you may always copy a name". One of our learners acquired the following somewhat more complicated actual conceptual model instead: "You may always copy a name unless the destination is the procedure identifier in a procedure call". The modification was due to his normal way of typing in the procdure identifier which he ended by hitting RETURN. The RETURN signaled the editor that the name was finished and that the actual parameters were next, so he got a menu asking for them. When copying the name he did not hit RETURN and so did not get this menu (since it was possible to modify the name after the copying, the editor did not automatically continue to the parameters after the copy operation). He did try to get to the parameters by selection. This worked but gave him another menu with some additional options, so he did not recognize it as the parameter-menu. After this he concluded that he could not copy the name of procedures. Since he was able to continue working with the only partly correct model, he did so for several days until he discovered the correct model.

5.4. Learners generalize from what they know

While the learners in the Mack et al. experiment generalized from their knowledge of typewriters, our learners generalized from their knowledge of traditional "glass-TTY" interfaces (interaction paradigm no. 2). For example, one learner was puzzeled that the editor as explained above did not show very long lines in full and proceeded to look through the user's guide for information on how to switch on the "word wrap" mode. The editor did not have such a mode and the user was supposed to use scrolling instead. But it took him a long time to realize this.

5.5. Learners have trouble following directions

Our learners certainly had the tendency to "jump the gun" and to try to do more than was expected in the step by step instructions for the self paced practice assignment. For example, when told to create an enumeration type with 15 elements and to put a "1" in the first element, many of our subjects carried on and also put a "2" in the second element and so on. This may be due to their unfamiliarity with the syntax editing concept of leaving placeholders unexpanded and return to them later. They felt uncomfortable about having an only partly finished program, but this gave them problems at later points in the practice instructions because their program did not have the presupposed contents.

5.6. Problems interact

Our learners were computer science students and professional programmers and did not have much difficulty with this aspect of learning to use a computer system. They were quite adept at isolating problems and not to carry on until they had solved each problem.

5.7. Interface features may not be obvious

Again, the interaction paradigm employed in a system implies a specific way of interpreting the interface features. But as long as users have not internalized the paradigm, they may have trouble with features that were considered obvious by the system designers and therefore are not explained in the user's guide or in the help system.

For example, EKKO has a window called the fragment window that can be in one of two modes, called 'survey' mode and 'detail' mode respectively. The name of the opposite mode of the current mode was always displayed as a soft function button that could be activated via the mouse to change mode. The concept of a soft functin button is not part of the earlier interaction paradigm known to our learners. One of our test subjects was able to carry out this mode shift operation but he thought that the mode name displayed was a status indicator and as such the name of the current mode and not of the opposite mode. He therefore formed the wrong mental model of the meaning of the mode names and would have had trouble if he had tried looking them up in the manual. He never did this however and so did not experience any trouble.

5.8. Help facilities do not always help

The EKKO editor does not include an online help system. The only help available was the user's guide which admittedly was rather poorly written. The test subjects practically did not use this guide at all after the first day. This may be because it was a poor guide, but it is likely also because the editor interface was so dynamic that further explanations were not needed once the underlying principles were understood. At the start of their learning period, our subjects had some problems using the manual because they were unfamiliar with the basic interaction paradigm used. They looked for information in the wrong sections of the manual because they were not aware whether some problem was described under the heading of general interface description or under some specific functionality.

5.9. Over-estimating the computer

Users have a tendency to feel awe for the computer. They do not have a clear understanding of exactly how 'clever' a given computer system is, and therefore they often will tend to believe that the system has more power that is actually the case. One should think that this problem would only manifest itself in naive users, but it turned out that some of our programmers showed the same behavior.

For example, when text was typed from the keyboard it would according to the user's guide be interpreted as either program text or comments according to "what makes sense". This was probably an unfortunate wording but it exposed a misconception on the part of most of our learners. They were not aware that "make sense" should be seen relative to the syntactical category of the place that was selected when the text was typed. Instead they believed that the editor would understand the text they actually typed in and place it as program text if it was in the form of legal program text and as a comment otherwise. In other words they empowered the editor with a capability it did not have.

6. Conclusions

Our programmers had learning problems because they were faced with three new issues at once. 1) They had to learn the commands of a new computer system. 2) They were not familiar with the functionality of the system in advance, so they had to learn that too. 3) The interaction paradigm through which they used the tool also was new.

Novice users who learn how to use a computer for the first time also have these three problems, so it is only natural that our programmers should manifest the same type of learning difficulties as novices do. Usually programmers and other experienced users are lucky and only have to face two or even only one of the three issues at a time.

Even though our test subjects as described had rather many learning difficulties they were nevertheless able to learn to use the editor in a few days. After this learning period they used the editor with great speed and without any serious trouble.

This experiment showed that programmers are able to move from using a text editor to a structure editor relatively easily even if they have not had training in the theory of formal grammers. An interesting problem would be whether programmers who had learnt programming using a structure editor would be able to make the reverse switch to a text editor with the same ease. This may not be the case. We could take driving automobiles having manual and automatic exchanges respectively as an analogy that shows that the switch is easy only in the one direction. Our conjecture is that the switch is easy when the system takes over functions that you had to do yourself before but difficult if you all of a sudden have to actively do something that the system used to handle.

Acknowledgements

The structure editor used in the experiment was programmed by Kurt Nørmark. He programmed the replay facility mentioned in section 4 as an addition to the editor, and he also actively took part in running the described experiment. The original design of the EKKO system was done by Karen Borup, Kurt Nørmark and Elmer Sandvad. This work was done with support from the Danish Natural Science Research Council (grant 11-4296). The Danish computer company Christian Rovsing A/S kindly provided the two professional programmers who participated in our experiment. Helpful comments on earlier versions of this paper was given by Micheal Mac an Airchinnigh and Brian H. Mayoh.

References

Borup, Karen; Nørmark, Kurt; and Sandvad, Elmer. 1983.
  EKKO - An Integrated Program Development System. Internal Report 51, Computer Science Department, Aarhus University.

Card, S.K.; Pavel, M.; and Farrell, J.E. 1984.
  Window-based computer dialogues. Proc. IFIP INTERACT'84 (London 4-7 September 1984), pp. 1.355-1.359 (in the preprint edition).

Mack, Robert L.; Lewis, Clayton H.; and Carroll, John M. 1983.
  Learning to use word processors: Problems and prospects. ACM Transactions on Office Information Systems 1, 3 (July 1983), pp. 254-271.

Neal, Alan S.; and Simons, Roger M. 1984.
  Playback: A method for evaluating the usability of software and its documentation. IBM Systems Journal 23, 1, pp. 82-96.

Nørmark, Kurt. 1985.
  Program development on graphical workstations. Proc. 18th Hawaii International Conference on System Sciences.

Zelkowitz, Marvin V. 1984.
  A small contribution to editing with a syntax directed editor. Proc. ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments (Pittsburgh, PA, April 23-25, 1984), pp. 1-6.

Author's present address

Jakob Nielsen, Computer Science Department, Aarhus University, Ny Munkegade, DK-8000 Aarhus C, Denmark.