DOCUMENT RESUME

ED 257 415 IR 011 600

AUTHOR Sleeman, D.; Gong, Brian
TITLE From Clinical Interviews to Policy Recommendations: A Case Study in High School Computer Programming. Study of Stanford and the Schools Technology Panel.
INSTITUTION Stanford Univ., Calif. School of Education.
PUB DATE Mar 85
NOTE 7p.
PUB TYPE Reports - Research/Technical (143)

EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS *Computers; *Computer Science Education; *Computer Software; Error Patterns; Interviews; *Programing; *Programing Languages; Secondary Education; Syntax
IDENTIFIERS *Debugging (Computers); Misconceptions; *PASCAL Programing Language

ABSTRACT

In order to determine the knowledge and skills needed by novice programmers to successfully learn computer programming, four studies were conducted using a clinical interview technique. The first study determined that many systematic errors in programming were due to programmers' high-level misconceptions of the nature of the computer and of the syntax and semantics of the programming language. The second study found that many misconceptions could be remediated effectively through a combination of (1) explicit training about the syntax and semantics of specific constructions in the programming language, (2) requiring learners to predict outcomes of short programs, and (3) providing students with interactive computer feedback. The third study examined methods used by high school teachers in computer programming instruction, and the fourth considered students' use of their existing knowledge. Findings indicate that, when Pascal programming functions are embedded with other functions, the embedded functions are evaluated incorrectly by novice programmers more often than when they are evaluated in their unembedded form. Results suggest a lack of a standard curriculum at the high school level and weaknesses in high school courses and textbooks in their treatment of debugging. Four references are listed. (LMM)

Study of Stanford and the Schools
Technology Panel

From Clinical Interviews to Policy Recommendations: A Case Study in High School Computer Programming

D. Sleeman and
Brian Gong

Stanford University School of Education

March 1985

One important question facing educational researchers today is "How can educational researchers, especially those in major research universities, develop more fruitful ties with high school policy makers and practitioners?" For the past two years, as part of a large, funded study called "The Study of Stanford and the Schools," we have been involved in conducting studies which have provided us several interesting research results and practical experiences which may be of value in formulating answers to the question posed above.

One division of the Study of Stanford and the Schools, the Technology Panel, has sought to investigate the impact that technology, primarily in the form of computers, is having on public education. While other members of the Technology Panel have studied the social impact of computers in the schools, our research has focused on understanding and improving the process of teaching and learning computer programming from a cognitive science perspective. We will first summarize some of our research in terms of questions, procedures, and findings. In doing so, we will note how the research evolved in conjunction with the cooperating teachers. We will then address, somewhat speculatively, some implications this research might have for improving instruction of computer programming in terms of classroom practices and school curriculum.

First, we will summarize our research.

I. Research

A. Research Questions

The Technology Panel's initial surveys of computer programming teachers confirmed that one of the teachers' primary concerns was how to better help their students learn. This concern helped shape our research questions. The fundamental question which has motivated our research is, "What knowledge and skills does a novice programmer need to develop to successfully learn to program a computer?"

Subsidiary questions include: "What misunderstandings do students commonly have in this subject domain?"

"Is it possible to remediate these misunderstandings?"

"How do these findings suggest students' classroom instruction be modified?"

Cognitive scientists have carried out studies comparing the organization and use of knowledge by "novices" and "experts" in several areas, including Mathematics, Physics, Language Arts, and Medicine. From these several studies we are forming an interesting overall impression, namely that "older" knowledge is less susceptible to change than recently acquired knowledge, in part because the student resists assimilating new knowledge if it conflicts with, or is counter-intuitive to, his earlier "commonsense knowledge" of the same domain. For example, di Sessa (in Gentner & Stevens, 1983) has shown that even engineering undergraduates may have an Aristotelian view of motion, which interfers with their understanding of Newtonian mechanics.

To follow up these questions, we have conducted a series of four studies. We have concentrated our investigation on the knowledge state of the learner because we are convinced that a detailed understanding of the learner can be very valuable in guiding instruction. However, while knowing what a student understands is desirable, acquiring this information can be difficult. We have patterned our research methodology after the clinical interview technique pioneered by Piaget, using a combination of highly specific diagnostic test items, protocol analysis, and interviews.

B. Research Results

Our first study established that many systematic errors in programming were due to programmers' high-level misconceptions of the nature of the computer and of the syntax and semantics of the programming language. An example of a misconception, the "semantic READ bug," is given below. When presented with the following BASIC program fragment,

```
10 READ LARGEST
20 DATA 2 9
30 PRINT LARGEST
```

a student with a semantic READ bug would say that the program would output "9" rather than the actual value "2." In a protocol-interview, the student might explain the action of this program fragment as, "In Line 10 the program asks for the largest number, so the computer will go through the numbers and pick out the biggest one. The largest number in the DATA on Line 20 is '9.' So in Line 30, where it says to 'PRINT LARGEST,' it will print '9'." This misconception reflects an erroneous conception of the computer as a machine that processes data semantically, instead of sequentially.

In our first study we worked with two schools to pilot the procedures. We then ran the final studies with three classes learning BASIC and two classes learning Pascal. In all, five high schools were involved. Over 75 students were individually interviewed.

From this study (Putnam et al., submitted), we identified over 30 distinct bugs in our sample of BASIC programmers. We clustered these misconceptions into eight classes or types, according to the BASIC syntactic concept with which the misconception was associated (Assignment statements, PRINT statements, READ statements, Variables, Loop construction, IF statements, Other flow of control difficulties, and Tracing and Debugging.) Similar results were found for high school Pascal Programmers (Sleeman et al., submitted).

In each case we provided the class teachers with detailed analyses of their students' performances. We have been told by several of the teachers that these detailed analyses of their students' misconceptions helped guide revisions of their class materials.

The students' misunderstandings having been diagnosed by means of the approach described above, the teachers with whom we worked independently and spontaneously moved to correct the students' errors. We adopted the teachers' concern as a research question, and have recently been exploring how it is possible to remediate the students' misunderstandings--if it is possible at all. (This is a real question, as researchers have reported that misunderstandings in other subject areas are frequently resistant to remedial treatment.)

Our second study indicates that it is possible to remediate many misconceptions very effectively by using a combination of explicit training about the syntax and semantics of specific constructions in the programming language, requiring the learner to make predictions about the outcome of short programs, and providing the student with interactive feedback from the computer.

In the second study we compared the pre- and post-remediation scores for students who received this type of remedial tutoring with a comparable control group of students learning BASIC. The remediated group improved significantly more than the control group, although some misconceptions appear to be more difficult to remediate than others. In general the students who received remedial tutoring for a particular topic did not make errors with the same topic in the post-test. In addition, several students appeared to spontaneously correct some of their misconceptions when presented with tasks and feedback which focused their attention on their misconceptions, i.e., they did not need explicit instruction.1

Our work on remediation thus far has been limited to one teacher at a sixth high school. This teacher was given detailed analyses of the errors made by his students and our recommendations to explicitly teach debugging skills. The teacher has indicated that he will incorporate our suggestions into his course. Additionally, we are discussing with him the more radical departure of teaching programming by giving students many more programs to read for the first half of the course.

A third study associated with our efforts has been to collect detailed information about how high school teachers actually teach selected topics in computer programming. This

research is providing a valuable context in which to interpret the misunderstandings identified in the first study. In addition, it will allow us to make more specific recommendations to teachers regarding their curricula and classroom practices.2

A fourth study, currently in progress, considers how the student uses the knowledge which he has. We are investigating human information processing in the context of computer programming by examining the relation of errors to cognitive processing demands in terms of the complexity of programming constructions. A principal finding of this study is that when Pascal programming functions are embedded within other functions, the embedded functions are more frequently evaluated incorrectly by novice programmers than when evaluated in their unembedded form. This result may be explained as novice programmers' processing errors made because of loss of information due to limitations of short-term memory, or "cognitive overload."3

In summary, four significant research contributions resulting from our recent work include: a) the extention of a suitable methodology for investigating knowledge representations and cognitive processes, b) the identification of common programming misconceptions, and construction of instruments and procedures to facilitate the diagnosis of such misconceptions, c) initial development of effective remedial training based on our catalogue of known "bugs," and d) investigation of the relationship of "cognitive overload" to errors in computer programming.

Our future work in this area will principally involve a more detailed remediation study to see which of the several components is more effective, and whether the remediation could be effectively given in a group setting as opposed to an individual student. Secondly, we wish to make a de ailed study of how students debug programs--when t y do--to determine what information they have about these techniques and to understand how debugging knowledge interacts with their knowledge of the programs' shortcomings.

II. Implications of Research

There are several implications suggested by our research. We will briefly touch on several points regarding both curriculum and classroom practice.

A. Curriculum

1. There is no standard curriculum at the high school level. Several of the misconceptions we observed may have been compounded because of sequencing or selection of topics. For instance, we have observed curricula that focused on applications programming (e.g., formatting business letters and spreadsheet output), mathematical programming (e.g., computing approximations to pi using various methods), and computer science topics (e.g., sorting techniques). Not only is there a lack of consensus of what should be included in a curriculum, there is disagreement about sequence and emphasis of topics.

This observation, of course, merely notes the existence of

an ongoing controversy of what constitutes an appropriate high school curriculum in "computer education."

2. In particular, we note that high school courses, and most textbooks, are particularly weak in their treatment of debugging. While debugging is a critical, common activity in computer programming, it is notable in its near absence from the high school curriculum. Our work in remediation of errors highlights the importance of strategies and skills to detect and correct errors. (More will be said about this in terms of classroom practice.)

B. Classroom Practice

1. Perhaps the most surprising observation of classroom practice coming from our research is the fact that a sizeable number of students in a class--as many as 25%--may have fundamental misconceptions of basic concepts, such as the "semantic READ," and the presence and nature of those misconceptions may go undetected (by teacher and students) for a whole semester.

2. This indicates that the instructional practice is not serving the students as well as it could. The simplest cure might be to make the teacher aware of common misconceptions, and provide materials to emphasize correct understanding of concepts most susceptible to misunderstanding. Indeed, we found that teachers readily adapted to handle misconceptions they were aware of. A second, easily implemented but less than comprehensive measure is to make available diagnostic paper-and-pencil tests or software that could quickly pinpoint common misconceptions. At the urging of the teachers with whom we worked, we are developing a prototype of such diagnostic software. This will be especially valuable to teachers, who lack the time and perhaps the training to conduct individual protocol analyses of their several students. A third measure is to provide more structure and practice in debugging for students, since many students spontaneously correct their misconceptions, given appropriate tasks and feedback. A more radical solution would be to emphasize more strongly the understanding of the "conceptual machine" and the programming language, by first having the students gain a firm skill in reading programs, and only then going on to producing, or writing, programs. Several workers have argued the virtue of having students study examples of programs which contain commonly used coding "chunks" (e.g., Sheil, 1981).

We anticipate a continued fruitful relationship with the schools as we share in the pursuit of better understanding and teaching of computer programming.

Footnotes

1 Contact D. Sleeman or Brian Gong, Stanford School of Education, for further information regarding this study.

2 Contact Jill Baxter, Stanford School of Education, for further information regarding this study.

3 Contact Laiani Kuspa or D. Sleeman, Stanford School of Education, for more information regarding this study.

References

Gentner, D. & Stevens, A.L. (Eds.). Mental Models. Hillsdale, NJ: Lawrence Erlbaum Associates, 1983.

Putnam, R., Sleeman, D., Baxter, J., & Kuspa, L. A Study of Misconceptions of High School BASIC Programmers. (submitted)

Sheil, B.A. Coping with Complexity. CIS-15. Palo Alto, CA: Xerox Palo Alto Research Center, 1981.

Sleeman, D., Baxter, J., Putnam, R., & Kuspa, L. Misconceptions in High School Pascal Programmers. (submitted)