USING SYSTEMATIC ERRORS TO INVESTIGATE THE DEVELOPING KNOWLEDGE OF PROGRAMMING LANGUAGE LEARNERS

Judith Segal[1], Khurshid Ahmad[1] and Margaret Rogers[2]

University of Surrey,
Guildford, SURREY GU2 5XH

[1] Centre for Information Technology Research, Department of Mathematics
[2] Dept. of Linguistic and International Studies.

Abstract

Inspired by recent research into second natural language acquisition, we investigated the systematic errors made by a large group of programming language students over a period of time. The major objective was to study the development of the students' knowledge. In this paper, we discuss one aspect of that development. The students had major difficulties with using the semicolon, the sequencing operator of the programming language ALGOL 68. We suggest that this is due to the fact that students did not immediately understand a specific simply stated rule of syntax, introduced in a decontextualised way, but rather that their understanding of the rule developed with their increasing experience of using it in different contexts.

1. Introduction

This paper describes the use of a learner-centred longitudinal study to illuminate how knowledge of one aspect of a programming language develops. We claim that a learner, when presented with a seemingly simple syntax rule, does not always immediately and completely assimilate the rule. On the contrary, we argue that, in general, the learner's understanding of the rule, depending on an understanding of the terms involved in the expression of that rule, develops as the rule is met in an expanding range of contexts. This is in contrast to the assumption of many teachers that when a simple syntax rule is taught, it is learned, and any errors made in applying it thereafter are the result of careless lapses. Evidence for our claim is based on a study of systematic errors made by the same group of learners over a period of time.

Recent work in the field of natural language acquisition (NLA) inspired both the longitudinal study from which our results are derived, and the use of a learner's systematic errors to illuminate her cognitive processes. In section 2, we discuss why NLA studies are of interest to researchers investigating the learning of programming languages, and the role of errors in such studies. Section 3 considers research into programming language learning (PLL). In recent years, PLL researchers have used errors as evidence of language-using and language-learning strategies, but, in the absence of any completed large-scale longitudinal studies, have not considered developmental issues. In section 4, we describe the large-scale longitudinal study carried out at the University of Surrey. The evidence it provides of learners gradually widening their understanding of the scope of a rule of syntax is presented in section 5. Conclusions are presented in section 6.

2. Natural Language Acquisition Studies

The relevance of NLA studies to PLL researchers[1]

There are obvious analogies between natural and programming languages. Both are instruments of communication and both involve the use of vocabulary and rules of syntax and semantics. In addition, the programming community has come to realise that not only does the efficiency of a program depend on making best use of machine resources, which was perhaps the main focus of attention in the 1960's and 1970's, but also on being comprehensible to people who may want to use, extend or adapt it. Soloway & Ehrlich (1984), Soloway (1986) and Joni & Soloway (1986) have pointed out that expert programmers use implicit rules of programming discourse, such as giving variables names which reflect their function, and that the violation of such rules makes programs very difficult to understand. This situation is analogous to that of natural language communication which also functions most efficiently when certain conventions are observed by the participants. These similarities suggest that researchers into PLL might profit from an examination of studies of natural language learning. Studies of second natural language learners are of particular relevance: learners of both second natural languages and programming languages have experience of learning and using at least one other language, the mother tongue.

Natural Language Acquisition Studies and the role of errors

[1]*The distinction between the two terms 'acquisition' and 'learning' is somewhat blurred: in NLA studies 'learning' has been taken to mean as occurring within a more formal framework than 'acquisition'. In this paper, we shall use the two terms synonymously.

The current view of the development of a child's first language is that it is a continual process in which the child tries to make sense of the language by actively developing a rule-base for deploying the language's vocabulary. The same process is considered to occur when a second natural language is acquired: the learner actively constructs a rule-base which develops towards the rule-base of the target language (see, e.g. Ellis (1986) for an overview).

In order to consider developmental issues, it is necessary to conduct longitudinal studies, in which the same group of learners is investigated over a period of time. The importance of longitudinal studies has not always been fully acknowledged. In the 1970's, researchers into second NLA, investigating the order of acquisition of certain language constructs, attempted to use cross-sectional studies, arguing that the more accurately a certain construct was used, the earlier it had been acquired. Rosansky (1976) presented evidence to refute this argument. She argued that "it would be far more profitable albeit more costly and time consuming to undertake more detailed longitudinal investigations of second language acquisition" (1976:424).

One way of gaining insight into the developing rule-base of the learner is through a study of her systematic errors. Before the seminal paper of Corder (1967), there was a tendency on the part of teachers to lump all instances of incorrect language usage together. Systematic errors, in which the incorrect usage matches the user's intentions, were not distinguished from unsystematic slips, in which usage does not match the user's intentions. Corder (ibid) argues that systematic errors may offer insights into the current state of the learner's developing rule-base, as well as into the strategies being deployed by the learner to use the language, and to organise and learn the rules of the language. For example, were a learner of English to say "they runned away" one might deduce that she has internalised a rule that past tenses by default end in -ed. The learner has presumably used (over)generalisation from examples to construct her rule. One could not make the same deduction from a correct usage, for example, "they walked away", as the learner may not be applying any internalised rule but may simply have remembered the whole phrase en bloc.

Besides generalisation, another learning strategy which may give rise to errors in second NLA (or PLL) is that of knowledge transfer, where an existing schema is adapted to a new situation. Here the learner already has knowledge of a language: such knowledge may be helpful, or may cause negative transfer errors, also called interference errors, when, for example, a syntax rule which is correct for the better known language is incorrectly applied to the new language.

3. Programming Language Learning Studies.

Terminology

We begin by explaining our terminology. As in the NLA studies, we shall aim to call instances of incorrect language usage 'errors' when they match the learner's intentions and 'slips' when such intentions are not matched. A learner should thus be able to recognise and correct a slip more easily than an error and slips will in general be made less consistently than errors. However, since the distinction between the two depends on knowing or inferring the learner's intentions, and since learner behaviour is rarely totally systematic, it is not always a straightforward matter to establish whether an incorrect usage is an error or a slip.

There is the added complication of bugs: the term 'bug' seems to be applied by computer scientists both to instances of incorrect programs or parts of programs and to misconceptions which will result in systematic instances of incorrect programs. Unfortunately, this ambiguous term, which we shall try to avoid, is very widely used in computer science literature.

PLL studies and the changing role of errors

Du Boulay and O'Shea (1981) review the literature on novices learning programming languages published in the 1970's. They report on several studies of learner behaviour, for example, studies of which language constructs are most used and which most prone to error. Errors are classified as syntactic, semantic, logical, clerical and stylistic. Of especial interest among the studies reported is that of Ripley and Druseikis (1978) into the 'errors' (which term the writers take to mean any instance of incorrect language usage) made by students learning Pascal. The writers were primarily interested in comparing the efficiency of different compilers and investigated the errors made by student programmers in order to build up a database of typically erroneous programs. Their study did not distinguish between errors and slips, probe the causes of errors or make any allowance for the fact that the nature of the errors made may depend on the level of experience of the programmer. Errors were classified by their surface characteristics into single token (missing, incorrect or extra) and multiple token errors. This study is of interest both because it is one of the few such empirical studies published at the time and because Pascal belongs to the same family of programming languages as ALGOL 68, the language of our study.

In the 1980's, studies of errors made by programming language learners have become less behaviouristic and more learner- as opposed to language-centred, as researchers, like those involved in NLA, have sought to investigate the possible

causes of errors in terms of the learner's own using and learning strategies. Among such studies are those of Jones (1982), Bonar & Soloway (1985), Du Boulay (1986), Pea (1986), Sleeman et al. (1986) and Spohrer & Soloway (1986). These studies are mainly concerned with the learning of procedural languages such as Pascal, but many types of errors might be independent of the programming language being learnt.

Learners often appear to attempt to repair gaps in their knowledge of programming language usage by referring to natural language usage, in particular to the assumptions made in natural language communication regarding the world knowledge and intentions of the other actors in the communication (see, for example, Bonar & Soloway, Pea, Du Boulay, and Sleeman et al.). Errors which may be accounted for by misapplied learning strategies are common: overgeneralisation (also called misgeneralisation) is recognised as a common source of errors, as is negative transfer. Confusion between natural language semantics and programming language semantics is commonly noted, for example, using THEN (part of the Pascal IF-THEN structure) to introduce a continuation step (Bonar & Soloway), or expecting the Pascal WHILE <condition> to continuously test for the condition rather than just once on each entry to a loop (Du Boulay). There are also errors which may be accounted for by negative transfers from mathematics, the system command language and, of course, other programming languages. Another source of errors is inadequate or inappropriate mental models of the computer or of the semantics of the programming language constructs (see, for example, Jones, and Du Boulay).

Opinion is divided as to whether language syntax does or does not cause big problems for novices. Presumably, this depends on such factors as the design of the language and the novice's previous experience with formal languages. Spohrer and Soloway (1986), investigating only syntactically correct programs written by university undergraduates, found that misconceptions regarding language constructs did not pose major problems. What did cause great difficulty was fitting the pieces of the program together i.e. composing the plans, 'plan' being the term given to a stereotypical block of code achieving a standard goal. The results of our study, as reported in section 5, demonstrate an analogous situation when considering syntax errors: students have great problems in fitting language constructs together.

Developmental Issues

We do not know of any instance (except that which will be described later in this paper) of a study of systematic errors being used to investigate the developing knowledge of the PL learner. As is pointed out by Leventhal & Instone (1988), although there have been studies investigating the separate characteristics of novices and experts, and the differences between them, "little research has

focused on the process of becoming an expert" (1988:1). Leventhal & Instone describe a pilot study and proposal for one large-scale longitudinal study of programming language learners to be conducted at Bowling Green State University investigating approximately 60 computer science students learning Pascal over the course of four years. It is proposed that in the course of the study, tasks very similar or identical to those which in the past have successfully demonstrated the differences between novices and experts, will be presented to the students at intervals under carefully controlled conditions.

4. The Surrey Longitudinal Study

The learner-centred longitudinal study, begun in 1987 at the University of Surrey, from which the data to be presented in section 5 were collected, was much more flexible in both its aims and operation than that proposed by Leventhal & Instone. Task-centred studies, in which data are collected under carefully controlled conditions, provide valuable information. However, in order to design such tasks, one must first have a hypothesis for the task to test. The tasks in the Leventhal & Instone study appear to be concerned with charting the diminishing difference between developing learners and a prototypical expert. We chose to focus on the learner rather than on any abstract expert, and had no preconceptions as to how the learner's knowledge might develop.

Our learner-centred longitudinal study (described fully in Ahmad et al. 1988) investigated 100 undergraduate engineering students (aged eighteen plus) learning the procedural programming language ALGOL 68. Our overall aim was to build as complete a picture as possible of the programming language learner. The aim of that section of the study with which this paper is concerned was, following the precedent set by the second NL researchers, to examine the errors made by the learners over the period of the study with a view to gaining some insight as to how their PL knowledge was developing.

The course consisted of one lecture a week over a period of two years beginning in 1987. Students could gain "hands-on" experience whenever they wished theoretically: in practice, "hands-on" practice was limited by a system of rationing the number of submissions made to the compiler and by practical difficulties caused by the number of potential users of the system. The students had all gained high scores in mathematics and science subjects in High School, and 90% of them had some previous experience of programming. This latter experience invariably involved BASIC: for 60% of all students, BASIC was the only high-level programming language they had encountered; the next most common procedural language was Pascal, of which 20% of all students had some experience. Over 40%

of all students had previously studied a programming language formally in a course leading up to a public examination.

Data Collection

We collected data by a variety of means: a copy of each program submitted by every student to the compiler was kept; questionnaires which included questions on comprehension and debugging as well as on the students' perceptions of the course were handed out and returned at each lecture; the investigator observed every lecture, 23 in the first year and 21 in the second, and the lecturer was interviewed as to his aims and objectives.

The learner-centredness of the study made the multiplicity of methods of data capture essential, both in order to generate and then to test hypotheses as to how the learners' knowledge was developing, and to provide confirmatory evidence in the absence of controlled experiments. Data from neither productions (those programs submitted by the students to the compiler) nor questionnaires were 'clean', in the sense of being carefully controlled. The data from the productions were very 'noisy' (a typically erroneous program would have errors and slips of many different types); the programs were produced in response to the lecturer setting assignments, five throughout the year, which tended to concentrate on specific language constructs rather than on fitting the constructs together; and it became apparent as the course progressed and the assignments became more complex, that individual productions were not always the work of individual students. As to the data collected from the questionnaires, which generally concerned comprehension and debugging, the questionnaires were answered within the relatively informal conditions of a lecture hall and on a voluntary basis, though between 70% and 100% of those students present did return the questionnaire at the end of each lecture. However, using data from both productions and questionnaires in conjunction provided valuable information. Unlike the Leventhal & Instone study, we had no tasks prepared prior to the beginning of the study but began by collecting all the first few programs submitted by each student to the compiler and examining the type, frequency and persistence of errors. From this evidence and data collected on the questionnaires, we formed hypotheses as to how the students' knowledge was developing, and tested these hypotheses by presenting the students with prepared debugging and comprehension questions on the questionnaires. Answers to such questions were analysed immediately, and frequently gave rise to new or amended hypotheses which were again tested. Details will be given in section 5.

The study was organised in such a way that individual case histories can be extracted without difficulty. However, in what follows, we shall present results in the form of aggregate group rather than individual data. In the teaching situation of the study,

where one lecturer taught 100 students, it was the aggregate group that the lecturer addressed, and aggregate trends that he was interested in.

5. Evidence of development

In this section, we describe how the study of errors made over a period of time can offer insight into how a simple rule of syntax may not be completely assimilated when first presented. We offer evidence to show that understanding of the rule was enhanced as the rule was encountered within a widening range of contexts, and suggest that the learner's developing understanding of the nature of the terms used in defining the rule plays a crucial role.

We had no preconceived notions as to where students' difficulties would lie, but it quickly became clear that they encountered problems with the use of the semicolon as a sequencing operator. This finding would come as no surprise to Ripley and Druseikis, who in their 1978 study of student errors in Pascal, commented that "if there is one thing that seems to appear time and time again in the error sample, it's the semicolonl" (1978:235). They did not, however, attempt to account for this. The lecturer of the course which is the subject of our investigation also expected the students to have problems using semicolons. He attributed such errors to careless lapses, postulating that the students' experience of natural language usage had not prepared them for an obligatory form of punctuation. We shall demonstrate that the pattern of errors in omitting semicolons showed a systematic nature which cannot be explained by reference to mere carelessness.

The rule for using the semicolon as a separator

Probably the first control structure that a student meets when learning a procedural language is sequencing, i.e. how to arrange for the computer to execute statement one and then statement two and so on. In ALGOL 68, the sequencing operator is the semicolon. A rule for using the semicolon was introduced in the fourth lecture of the course viz:

"Given a;b where a and b are processes, ';' means 'wait until process a has finished and then start process b' ".

In the previous lecture, they were told that:

"a process is anything which processes data"

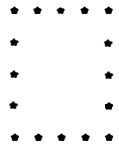
In other words, the semicolon in ALGOL 68 is used as a statement (process) separator, as in Pascal, with the difference that most Pascal compilers will accept empty statements, so, for example, putting a semicolon immediately preceding a construct terminator is acceptable in most versions of Pascal, but unacceptable in ALGOL 68.

We shall present evidence that the students did not immediately understand the rule and hence were not able to apply it in new contexts. We claim that what caused the students difficulty was understanding the range of the term 'process'.

The Empirical Evidence

The lecturer's intention was to concentrate initially on the flow of control through a program. He introduced the sequencing operator, the semicolon, in lecture 5, procedures in lecture 5, DO loops in lecture 7 and IF structures in lecture 10. Subsequent lectures dealt with data structures and other issues.

The lecturer set the first two assignments in lecture 5 and lecture 7.[2] The first assignment was to write a program which would print an empty box as output:



The assignment was designed to test the students' use of the sequencing operator, and of procedures, which they were required to use, and which were taught in lecture 5. A model answer is shown in figure one. As the lecturer's intention was to concentrate initially on control structures, he provided the students with a library of self-explanatory primitives viz print a star, print a space and next line (these primitives are predefined procedures). In the model answer (figure one), lines 3 - 7 are concerned with declaring procedures and lines 8 - 12 with calling (executing) them.

[2] It is perhaps worth recalling at this point that each year of the course consisted of one lecture a week for approximately 25 weeks. Thus, lecture 5 was in week 5, and so on.

```
     N.B. line numbers are for reference only
1.   PROGRAM assignment1
2.   (
3.   PROC spaces = VOID: (print a space; print a space; print a space);
4.   PROC star = VOID: print a star;
5.   PROC stars = VOID: (star; star; star);
6.   PROC outer line = VOID: (star; stars; star; next line);
7.   PROC inner line = VOID: (star; spaces; star; next line);
8.   outer line;
9.   inner line;
10.  inner line;
11.  inner line;
12.  outer line
13.  )
14.  FINISH
```

Figure one: Model answer for the first assignment.[3]

In order to investigate how this assignment was executed, 20 students were chosen at random, and all the submissions made to the compiler by each student were examined. Of those 20 students, 8, in at least one of their submissions, omitted the semicolon after the procedure declarations but not elsewhere. This was despite the fact that the students had all seen a program involving a procedure declaration followed by a semicolon, one of only two program examples presented up to this point of the course.

For assignment two, students were required to write a program which would print as output three stacked boxes of stars of size 7 x 7, 5 x 5 and 3 x 3 respectively:

```
* * * * * * *
* * * * * * *
* * * * * * *
* * * * * * *
* * * * * * *
* * * * * * *
* * * * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * *
* * *
* * *
```

There are many possible ways of tackling this problem using, for example, nested loops, taught in lecture 7, or procedures with parameters, taught in lecture 8. One possible model answer using nested loops is shown in figure two. Other answers, written by 11 of the 20 students investigated, involved the use of procedures. Omitting the semicolon after declaring a procedure was still a very dominant error,

[3] 'VOID' on lines 3 - 7 indicates that a call to any of the procedures declared in these lines returns no value to the program.

evidenced by the fact that it was made by six of the 11 procedure-using students. Of these six, five had made (and corrected) the same error on assignment one, where the sixth student had used the semicolon correctly. The most frequently made error was omitting the semicolon after OD when the loop occurred as part of a sequence (line 10 in figure two). 11/20 of the students made this error. (Note that a semicolon must not be placed after the OD on line 12 as it comes before a terminator of the outer loop, nor on line 13 which immediately precedes the program terminator).

```
        N.B. line numbers are for reference only
1       PROGRAM ass2
2       (
3       FOR n FROM 7 BY -2 TO 3
4.      DO
5.              TO n
6.              DO
7.                      TO n
8.                      DO
9.                        print a star
10.                     OD;
11.                     next line
12.             OD
13.     OD
14.     )
15.     FINISH
```

Figure two: a possible answer to assignment two.

In order to investigate further, students were asked to correct the following program as included in the questionnaire of lecture 12:

The following program is syntactically incorrect (as usual, the line numbers are included for reference purposes only):

```
1.      PROGRAM pattern
2.      (
3.      PROC stars = VOID: (print a star;print a star; print a star)
4.      PROC spaces = VOID: (print a space; print a space; print a space)
5.      PROC line one = VOID: (spaces; stars)
6.      PROC line two = VOID: (stars; spaces; stars)
7.      TO 5
8.       DO
9.         line one;
10.        next line;
11.        line two;
12.        next line;
13.      OD
14.     stars;
15.     stars;
16.     stars;
17.     )
18.     FINISH
```

(+) (The + and -
(+) signs in
(+) parentheses
(+) indicate
where, in a
correct solution,
semicolons
should be
inserted and
(-) omitted. Of
(+) course, this
information
was not
(-) given to the
students.)

Figure three: a debugging task set on lecture 12.

Here the results were that 78% of a population of 78 correctly inserted the semicolons after the procedure declarations on lines 4 - 6 but only 28% correctly inserted the semicolon needed after the OD[4] on line 13. (It should be noted that the semicolons occurring on lines 12 and 16 before the loop and program terminator also

[4]Recall that DO loops were taught in lecture 7.

constitute errors. However these errors are not relevant to this particular investigation.)

The hypothesis was formed that what was causing the students such difficulty in applying the semicolon rule was the word "process". They were happy that simple statements, e.g. print a star, were processes, but found it difficult to accept immediately that complex structures such as procedure declarations or loops were also processes. The results of the error analysis of the programs produced for the assignments spanning lectures 5 - 9 suggest that the students found it very difficult at this stage of the course to accept that a procedure declaration is an example of a process. Perhaps a procedure declaration which just adds an identifier to an environment is not conceived of as actively processing data. By lecture 12, the analysis of answers to the debugging task in figure three suggests that the students' understanding of the term 'process' had widened to include procedure declarations, but did not yet encompass a loop structure constructed from simpler processes.

In order to test the hypothesis that students did not yet understand that the term 'process' encompasses complex block structures, the students were presented with the following question on questionnaire 17, at lecture 17:

N.b. the line numbers are for reference only
The following program is syntactically correct EXCEPT that I have left out all the semi-colons:

```
1.  PROGRAM pattern
2.  (
3.  PROC pattern = (INT r) VOID:        ( FOR n TO r
4.                                       DO
5.                                         TO 3
6.                                         DO
7.                                            print ("a")                    (+)
8.                                            print ("b")                    (+)
9.                                            print ("a")
10.                                        OD                               (+)
11.                                       print (newline)                   (+)
12.                                       IF n = 2 THEN TO n
13.                                                          DO
14.                                                          print("a")       (+)
15.                                                          print (newline)
16.                                                          OD
17.                                       ELSE TO n
18.                                                          DO
19.                                                          print("b")       (+)
20.                                                          print(newline)
21                                                           OD
22.                                       FI                                (+)
23.                                       TO 3
24.                                       DO
25.                                         print ("aba")
26.                                       OD                                (+)
27.                                       print (newline)
28.                                       OD                          )     (+)
29.           pattern (3)
30.           )
31.           FINISH
```

(i) Please insert all the semi-colons missing above
(ii) Please draw below the output you would expect from the corrected program.

Figure four: Debugging question asked at week 17 of Year One.
(N.B. As in figure three, the lines at the end of which semicolons should be inserted are marked by (+)).

Semicolons were required at lines 7, 8, 10, 11, 14, 19, 22, 26 and 28 (after the closing parenthesis). If the hypothesis were correct i.e. that students found it difficult to accept that complex structures such as loops or selection constructs were in fact single processes, then, looking at the number of people who omitted semicolons on a line-by-line basis, one would expect peaks in these numbers at lines 10, 22, 26 and 28. The frequency of semicolon omissions on a line-by-line basis is given in figure five.

Frequency of semicolon omission

Line number	7	8	10	11	14	19	22	26	28
Frequency	4	4	17	8	4	5	24	27	31

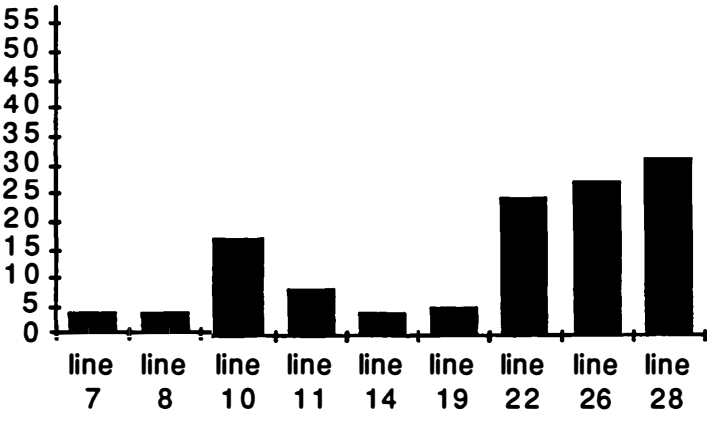


Figure five: frequency of semicolon omission on a line-by-line basis *Total number of students = 58*

The peaks of the frequency of semicolon omission are as predicted, at lines 10, 22, 26 and 28 i.e. semicolons are most likely to be omitted at the termination of some complex structure in sequence with other processes. This would appear to support the hypothesis that the students at this point of the course did not appreciate that complex structures such as loops are in fact single processes.

Corroborative evidence was obtained from a similar question asked of the students who entered the course in the following year, 1988, half-way through their first year. These students were of similar background to the ones investigated above and were taught by the same lecturer according to the same syllabus.

The question was:

. PLEASE ANSWER THIS QUESTION UNAIDED.

The following program is syntactically correct except that I have omitted all the semicolons:

```
1   PROGRAM test894
2   (
3   TO 4
4    DO
5      print a star
6    OD                                                    (+)   (As usual, (+)
7   next line                                              (+)   indicates those
8   FOR a TO 4                                                   lines at which
9    DO                                                          a semicolon should
10          print a star                                    (+)   be inserted.)
11          IF ODD a THEN
12                              TO a
13                               DO
14                                 print a star
15                               OD                        (+)
16                              next line
17                  ELSE
18                              print a star               (+)
19                              next line
20          FI                                             (+)
21          print a star                                   (+)
22          next line
23   OD
24  )
25  FINISH
```

Please clearly insert all the missing semicolons.

Figure six: a debugging question set in the middle of their first year for the 1988 entry students.

The lines at the ends of which semicolons are omitted are 6, 7, 10, 15, 18, 20 and 21. If the results of the entry of the previous year were to be replicated, we would expect that the greatest frequencies of semicolon omission would be on lines 6, 15, and 20. The results are shown in figure seven.

Frequency of semicolon omission

line number	6	7	10	15	18	20	21
frequency	12	5	9	16	3	25	2

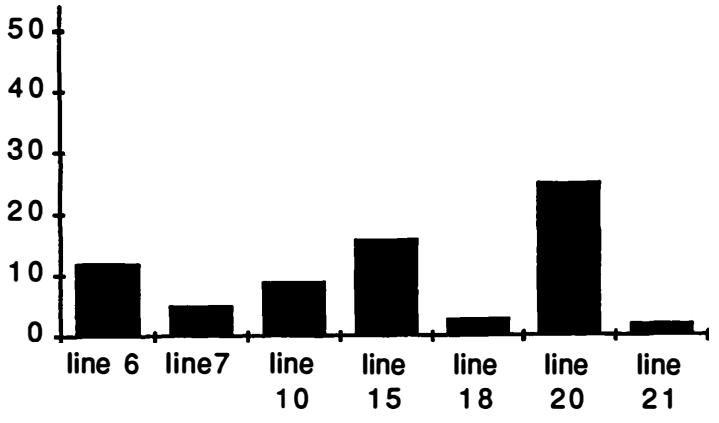


Figure seven: semicolon omission on a line-by-line basis (1988 entry). *Total number of students = 54*

The results as shown above show a similar trend to the findings of the investigations with the first group of students: the students tend to omit semicolons where a complex process is in sequence with other processes. There is further evidence of these dominant errors being the result of misconceptions rather than simple coincidence from the consistency of the errors. For example, all the students who omitted the semicolon after the OD on line 6 also omitted it after the OD on line 15.

In the second year of their course, the original (1987 entry) group of students were presented with the same debugging question on two of their weekly questionnaires: once in week 8 and again in week 20. The question is as shown in figure eight.

This program is syntactically correct except that I have omitted all the semicolons.

Please indicate clearly where the semicolons should be inserted. (N.B. the line numbers are for my reference only)

```
1.  PROGRAM test year 2
2.  (
3.  PROC pattern = (INT r)VOID:
4.          (
5.          PROC stars = (INT s)VOID:
6.                              TO s
8.                              DO
9.                                print a star
10.                             OD
11.         PROC spaces = (INT t)VOID:
12.                             TO t
13.                             DO
14.                              print a space
15.                             OD
16.         FOR n TO r
17.         DO
18.           IF ODD n THEN stars (n)
19.                                print a space
20.                               stars(n)
21.                   ELSE spaces (n)
22.                                print a star
23.           FI
24.           next line
25.         OD
26.         TO r
27.          DO
28.            stars(2*r + 1)
29.            next line
30.          OD
31.         )
32. pattern(3)
33. )
34. FINISH
```

Figure eight: Debugging question set in weeks 8 & 20 of year two of the 1987 entry.

The question was designed to be similar but not identical to the question asked in week 17 of year one. A correct solution should have semicolons inserted at lines 10, 15, 18, 19, 21, 23, 25, 28 and 31. If the misconception that complex structures cannot be single processes were to remain, one would expect dominant error frequencies at lines 23, 25 and 31.

The results are shown in figure nine, (week 8) and figure ten, (week 20). For the sake of comparison, we repeat figure five. Of the 77 students who answered at least one of the three questionnaires being compared (17 of year one, 8 and 20 of year two), 23 answered all three, and 25 answered two of the three.

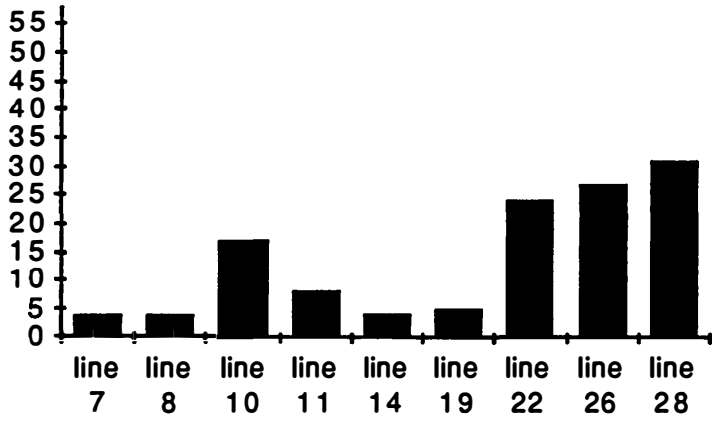


Figure five revisited: frequency of semicolon omission in response to question on questionnaire 17 of year one; *total number of students = 58*. (Predicted maximal error frequencies at lines 10, 22, 26 and 28).

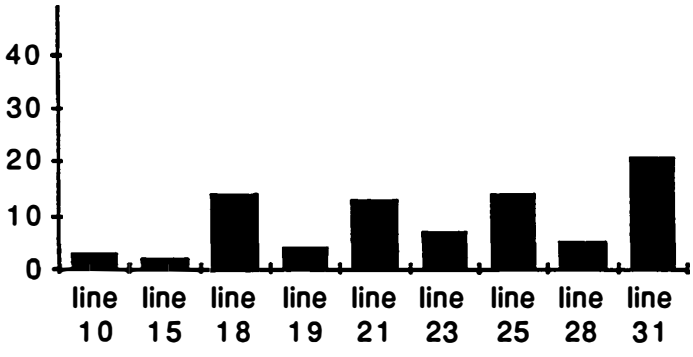


Figure nine: frequency of semicolon omission in week 8 of year two : *total number of students = 49*

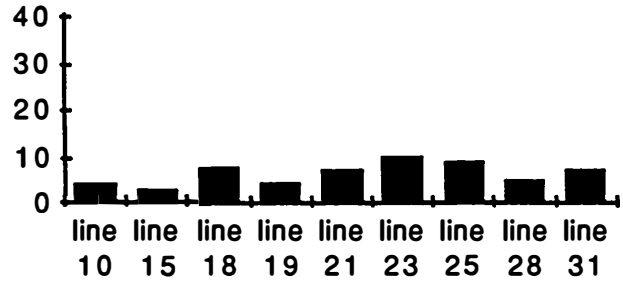


Figure ten: frequency of semicolon omission in week 20 of year two : *total number of students = 41*

In figures nine and ten, if the misconception remains that complex structures, such as

loops or IF structures, cannot be single processes, the predicted maximal error frequencies would be at lines 23, 25 and 31. Figure nine shows some of the predicted pattern, though not as pronounced as that of figure five.[5] What is particularly striking in comparing figures five, nine and ten, is how, as time progresses, and the students gain more experience, the charts of semicolon omission become flatter. The predicted pattern of dominant errors shown in figure five, half-way through the first year of the course, gives way to the almost flat distribution of figure ten, at the end of the course. There is no longer the marked dominance of certain errors in figure ten, indicating that the students' understanding of the term 'process' has widened to include complex looping and selection structures, and that their semicolon omission errors occur at random (and might thus be referred to as 'slips').

Having compared the overall pattern of the frequency of semicolon omission errors, we now turn our attention to specific constructs. The following table makes a direct comparison between similar instances of constructs in the debugging questions relevant to figure five and figures nine and ten (the questions are given in figures four and eight):

[5]As well as that predicted of lines 25 & 31, lines 18 & 21 in figure nine have a greater error frequency than the average . Questioning the students led to the hypothesis that this was due to problems of spacing: the students expected lines containing key (reserved) words (those in uppercase letters in this example) to contain **only** key words. Future research might investigate to what extent students are influenced by spacing in free-format languages such as ALGOL 68

	week 17 year 1	week 8 year 2	week 20 year 2
Population	58	49	41
[23 students answered all three questionnaires; a further 20 answered that of year 1 and one of those of year 2]			
Omission of semicolon			
(i) after complex proc. declaration (line 28 of fig.4, 31 of fig.6)	31 (53%)[6]	21 (43%)	7 (17%)
(ii) After DO loop (lines 10 & 26 of fig.4, 25 of fig.6)	17 (29%) 27 (47%)	14 (29%)	9 (22%)
(iii) After IF construct (line 22 of fig.4, 23 of fig.6)	24 (41%)	7 (14%)	10 (24%)

Table one: table of comparison of semicolon omission in similar instances in week 17 of year one, and weeks 8 & 20 of year two.

The table shows that considering semicolon omission after each construct in the left-hand column separately, the trend in the proportion of students who make such an error over the period is undeniably downward.

6. Summary and discussion

In this section, we consider the specific conclusion of the study, some directions for future research, and a discussion of wider issues.

We have presented evidence that students do not necessarily understand a simply stated syntax rule completely from the outset, as evidenced by their failure to use it correctly in new situations. Rather their understanding of the rule, depending on their understanding of the terms used in defining that rule, develops as they meet the rule in an expanding range of contexts. We suggest that what caused the difficulty in the

[6]The percentages given are the percentage of the population making that error.

instance considered in this paper was the use of the term 'process', defined in very general terms at the outset of the course (see section 5). The results reported in this paper demonstrate that only after much experience of usage, including at least two assignments, did the students' understanding of the term 'process' develop to include procedure declarations. The analysis of the replies to questionnaire 17 (figures four and five) indicate that at lecture 17, two thirds of the way through the first year, the students' understanding did not include complex language structures, defined in terms of other structures, within the compass of the term. By week 20 of year two, however, we can deduce from the uniform distribution of semicolon omission errors shown in figure eight that such errors occurred at random, which suggests that they were now the result of careless lapses rather than systematic misconceptions.

One immediate result of the study was to enlighten the lecturer about the nature of the students' difficulties with semicolons. In a real-life situation, it is not always clear to a lecturer how the students are progressing. In this case, because of the high (and unfortunately not uncommon) student-staff ratio (100:1), the students' interactions with the lecturer were limited to individual queries at the end of lectures and at other odd moments, and to the submission of 5 assignments throughout the year. As the assignments of necessity could not exhaustively cover the course content, and as there is considerable evidence that individual assignments were not always the work of individual students, they did not provide the lecturer with a very clear picture of the students' development.

Possible major directions for future research include developing an interactive system designed to automate the diagnostic capabilities of the questionnaires, and investigating whether the developmental nature of rule-understanding is replicated with other rules in other formal languages. The question as to whether students have the same difficulty in identifying single processes in related block-structured languages such as Pascal, might be difficult to resolve in the absence of a precise rule of syntax. In this study, it was evidence of errors made in the surface structure of the language, i.e. the omission of semicolons, which alerted us to possible misconceptions regarding higher-level language concepts, i.e. the block structure of the language. Where syntax rules are not so rigorous, for example, in those versions of Pascal which will accept empty statements and hence semicolons before construct terminators, such evidence may not be forthcoming. In such cases, the default in situations in which the learner is unsure as to whether or not a semicolon should be inserted, seems to be to put one in, as evidenced by the common error of inserting a semicolon in an IF-THEN-ELSE construction before the ELSE (see, for example, Du Boulay (1986)). In ALGOL 68, on the other hand, the default seems to be to omit the semicolon., and it was these omissions which provided us with the data for this study.

This paper has addressed wider issues than the study of a specific category of learners using a specific syntax rule of a specific language. We have argued that,

following the lead of researchers into natural language acquisition, researchers into programming language learning might find that learner-centred longitudinal studies offer much valuable information as to how the learner's knowledge develops. We have also argued that PLL researchers and teachers of programming languages should follow the lead of NLA researchers and teachers of natural languages in their attitude to errors. Systematic student errors should be regarded not as mere slips which must be remediated but rather as possible evidence of the student playing an active part in her own learning by deploying learning and using strategies, as described in the papers summarised in section 3, and as evidence of how the student's knowledge of the language is developing, as described in this paper.

Acknowledgements.

The authors would like to express their deep gratitude to Dr. B. M. Cook and Mr. R. M. A. Peel, of the Department of Electrical Engineering of the University of Surrey, and, of course, to the students who were the subjects of this study, for their very willing cooperation. We would also like to thank Mr. T. F. Goodwin, erstwhile Director of the Computing Unit at the University of Surrey, for making available to us the facilities of the Unit.

During the period of this study, Judith Segal was supported by a Research Fellowship provided by British Gas, under the auspices of the Women Returners Scheme, coordinated by Professor D. Jackson of the University of Surrey, and the Womens' Engineering Society.

References

Ahmad K., Segal J., Hogger E., Rogers M. (1988), "Learning about the Programming Language Learner: a prelude to ICAL", Technical Report, CITRUS/TR88.9, Centre for Information Technology Research. Department of Mathematics, University of Surrey, Guildford, Surrey, England.

Bonar J. & Soloway E. (1985), "Preprogramming knowledge: a major source of misconceptions in novice programmers", Human-Computer interaction, Vol. 1, pp 133-161.

Corder S.P. (1967), "The significance of Learners' Errors", International Review of Applied Linguistics, Vol 5, pp 161-170.

Du Boulay B. & T. O'Shea (1981), "Teaching novices programming" in Computing Skills and the User Interface, Coombs M.J. & Alty J.L. (eds.), Academic Press.

Du Boulay B. (1986) "Some difficulties of learning to program", Journal of Educational Computing Research, Vol.2, No. 1, pp 57 - 73.

Ellis R. (1986) "Understanding Second Language Acquisition", Oxford University Press.

Jones A. (1982), "Mental Models of a first programming language", CAL Research Group Technical Report no.29, The Open University, Milton Keynes, England.

Joni S-N & Soloway E. (1986), "But my program runs! Discourse rules for novice programmers", Journal of Educational Computing Research, Vol.2, No. 1, pp 95 - 125

Leventhal L. & Instone K. (1988), "Becoming an Expert?: The process of acquiring expertise among highly novice computer scientists", Pilot & Proposal, 02-SEP-88, Dept. of Computer Science, Bowling Green State University, Bowling Green, Ohio, U.S.A.

Pea R. (1986), "Language independent conceptual "bugs" in novice programming", Journal of Educational Computing Research, Vol.2, No. 1, pp 25 - 36.

Ripley G.D. & Druseikis F.C. (1978), "A Statistical Analysis of Syntax Errors", Computer Languages, Vol.3, pp 227 - 240.

Rosansky E. (1976), "Methods and Morphemes in Second Language Acquisition Research", Language Learning, Vol. 26, no. 2, pp 409 - 425.

Sleeman D., Putman R., Baxter J., Kusper L. (1986), "Pascal and High School Students: a study of errors", Journal of Educational Computing Research, Vol.2, No. 1, pp 5 - 23.

Soloway E. & Ehrlich K. (1984), "Empirical studies of programming knowledge", IEEE Transactions on Software Engineering, SE-10:5, pp 595-609.

Soloway E. (1986), "Learning to program = learning to construct mechanisms and explanations", Comm ACM, Vol.29 no.9, pp 850 - 858.

Spohrer J. & Soloway E. (1986), "Novice mistakes: are the folk wisdoms correct", Comm. A. C. M., vol.29, no.7, pp 624 - 632.