Learning programming: an Analysis of Looping Strategies Used by Beginning Students

RENAN SAMURÇAY

1. Introduction

The relationships between education and computers cover a very large domain of investigation [Pea, 1984]. We think that it is necessary to distinguish between two main domains, each characterized by the way the computer is used

— the computer is used as a medium for acquiring knowledge other than programming concepts, like mathematics, physics, etc. ...
— the computer is used to educate students in its use.

In the former case (for example, in the use of educational software), the user does not have to know programming. It may be sufficient for him to know how to use the software by typing on specific keys of the computer.

Our own work is concerned with the second domain. The main questions for us are:

- the identification and analysis of conceptual difficulties encountered by students who are learning programming;
- the identification and analysis of didactical situations which can permit the teacher to simplify the acquisition process.

In this paper we present a preliminary study which approaches these questions.

As revealed by many recent researches [Pea and Kurland, 1983, 1984, Soloway *et al.*, 1982, Bonar, 1984] programming, even at a simple level, is a difficult activity to learn. The difficulties of mastering the use of computers (new technological tools) come not only from their practical aspect (utilization), but more particularly from their theoretical aspect (the representation of programming concepts). We are interested in the transmission of this complex knowledge within the school environment.

2. Identification and analysis of meaningful programming concepts for beginning students

Programming activity covers a very large field of concepts and procedures, like variable, assignment, looping, recursion, procedure*, function*, data structuring, etc. These concepts and procedures are not all on the same level of complexity. It is possible to establish a conceptual hierarchy between them. When we talk about a conceptual hierarchy or a conceptual analysis, it is very important to understand that this analysis is based on *both* an analysis of the subject's activity *and* an analysis of the subordinate concept. The analysis of concepts is not reducible to one of them. [For the definition of "concept", as used here, see Vergnaud, 1982].

For example, if we analyze the concept of procedure (in the sense of a sub-program), we can see that its comprehension requires the mastery of the concept of variable (the concepts of *local* and *global* variables make sense in the context of procedures). Data structuring—one of the most important concepts of programming which is not introduced in our didactical experiment on account of its complexity—requires not only the concepts of variable, looping, etc., but also some concepts directly related to the modelization of the given problem (for example, files, lists and representations in tree form). We can also establish a hierarchy between the three types of loop presented by the PASCAL language (*for-*, *repeat-* and *while*-loops). For example, the *while*-loop necessitates a plan of action of the form

test variable/process variable

which is more "natural" than the form

process variable/test variable

involved in the *repeat*-loop [Rouchier, Samurçay, 1984]. We can also hypothesize that the *while*-loop is conceptually more difficult than the *repeat*-loop.

One of the main questions for research on didactics is the determination of the conceptual field around which the didactic experience will be organized. A given situation does not involve just one concept, nor all the properties of one concept. This is the reason why the idea of a conceptual field, developed by Vergnaud [1982], is so powerful. In other terms, the question is which concepts and procedures must be introduced in order to solve a significant class of problems.

When one examines programming activity—in particular with a compiled programming language—one easily remarks that the concept of variable is at the center of every problem situation, however simple it is. Thus it is easy to justify the decision to introduce the concept of variable rather quickly. But if one thinks about the set of situations in which the programming concept of variable makes sense, it is necessary to take account also of the concepts of loop and of assignment.

It is not possible to study these concepts separately

*i.e. procedure and function as programming concepts because they are tightly connected with one another. The analysis given below shows more precisely the relationships between these concepts. And the reader will understand why we organized our introductory course around problems requiring the looping construct.

3. Brief analysis of the conceptual field involved in the beginner's programming activity

In this section we will analyze the three principal concepts of variable, loop and assignment which are involved in our didactical experiment. This analysis, as we have said above, takes account of the meaning of these concepts for the student, i.e., the analysis of different situations in which the student encounters the same concept but in each case with a different significance.

We consider that repetition is one of the most important functions of computers. One can repeat a very large set of actions, as many times as one wants. Moreover the concepts of variable and assignment achieve their full meaning in the looping activity. With respect to their significance for the student, we distinguish four forms of occurrence of the assignment sign. This distinction is not made in programming theory itself because structurally all these forms are equivalent.

1. Assignment of a constant value

```
A :=(*) 3
list := word
test := false
```

2. Attribution of a calculated value

```
A := 3+5
B := 3*K
```

3. Duplication

```
K := L
```

4. Accumulation

```
sum := sum + number
expox := expox*x
x := x+5
```

It is obvious that the only case in which the notions of variable and assignment take their full "programming" sense is the last one: accumulation. The student has to designate by the same name both the preceding value and the present value, which is a function of the former, and to treat the assignment sign as an asymmetric relationship.

In the other cases the student may use his existing mathematical conception of variable and equality, i.e., to the name of a variable can be associated a unique value.

The looping strategy involved in the problems we propose to the students is the *repeat*-loop. A *repeat*-loop has the following general structure:

```
initialization
    repeat
        actions
        until  condition
```

(*) We use the PASCAL notation for the assignment sign.

It is formed of three elements:

— *the loop-invariant:* a block of actions which is repeated.
— *the test:* a condition for terminating the loop
— *the initial state* of variables to be transformed in the loop.

One can see that every looping activity also requires different variables to be operated on.

We call a "variable-plan" the set of operations that the student has to consider in order to construct a correct looping strategy. These operations are

- *description* (What is the function of a particular variable in the program?)
- *update* (How is it used, what is the law of succession?)
- *initialization* (How is the initial value given?)
- *declaration* (What sort of variable is concerned? an integer, a real number, or a structured type of data?)

We assume that expert programmers follow the above plan when solving problems. The question is how to characterize the plans used by beginning students. One can hypothesize that the plans used by beginners will be based on previous knowledge and familiar procedures that relate to the specific problem to be solved [Samurçay, Rouchier, 1984]. For example, it is possible for the students to use their mathematical model of a variable. But this conception is insufficient because the mathematical description of a variable is a static description, i.e. it designates the representative of a set

$x \in [a, b]$

or an unknown in an equation

$x + 2 = 5$

The description of a programming variable is more dynamic; the student has to identify the law of transformation of its value, as, for instance:

$n := n + 1$

Although all variables used in a program are equivalent from the point of view of computer science, their cognitive significance differs for the user. For example (see Figure 5) the variables *number, sum, counter,* each have a different status. *Number* represents values to be read, *sum* and *counter* represent calculated values. Moreover the activity of counting and the activity of accumulating are different. Whereas counting is naturally represented as a "successor function"

```
counter := counter + 1
```

i.e increasing the previous number by 1, the update of the accumulation variable

```
sum := sum + number
```

requires the addition of another variable. When faced with developing an assignment for an "accumulation variable", students must really confront their understanding of the particular type of assignment statement needed in a particular content.

4. Analysis of problems

We examine now the five problems we proposed to the class (see Figures 1 to 5). While these problems are relatively simple, their solutions illustrate many of the basic notions of programming. They all use a few repetitive actions involving simple arithmetic operations. As we indicated above, for each problem the student has to elaborate a variable-plan for each of the variables involved.

Figure 5 shows the complete solution and the different variable-plans associated with problem 5. For each of the other problems we only give the loop-construction part of the solution. The reader can easily construct the variable-plans by following the model of example 5.

```
P1 - Write a program which calculates and prints out the value of the
     expression

        (x+y)^n      x y n   can be any integers.
```

```
a := (x+y)
count := 1
    repeat
      a := a*(x+y)
      count := count + 1
    until count = n
```

Figure 1

In the first problem the principal cognitive task involved in the solution is the construction of two variable-plans for the accumulation variable *a*, and for the counter-variable *counter* respectively. This construction implies the representation and the procedural expression of "$(x+y)^n$ is obtained by multiplying repeatedly $(x+y)$ by itself *n* times; the repetition terminates when *n* multiplications are done".

```
P2 - Write a program which calculates and prints out the value of

        x^n+y^n     x y,n  can be any integers
```

Figure 2

The second problem is similar to the first; x^n and y^n must be calculated following the above procedure and then added. The student has to construct three variable-plans: one counter-variable plan and two accumulation-variable-plans for the calculation of x^n and y^n.

```
P3 - Write a program which calculates and prints out the sum of the
     first  n  integers
```

```
   sum := ϕ
number := ϕ
       repeat
          number := number + 1
          sum := sum + number
       until number = n
```

Figure 3

In the third problem the student has to construct two *related* accumulation-variable-plans, i.e two laws of succession for the variables *number* and *sum*. Notice that though the variable *number* occurs like a counter-variable (i.e., the successive values are obtained by increasing the previous number by 1), it is not a counter-variable as regards its description: it permits the successive terms of the series of integers to be generated. Thus it can be considered as a new-value variable (see Figure 5). So the student has to represent two related aspects:

— the *number* added to the sum is obtained by adding 1 to the *previous* number;

— the *sum* is obtained by adding to the *previous* sum the number obtained in the above manner.

Notice that the order in which the *number* and the *sum* are obtained is important: the conditional statement should test the *number* still to be added.

```
P4 - Write a program which calculates and prints out the average of 5 (and
     then 40) integers 'read in'.
```

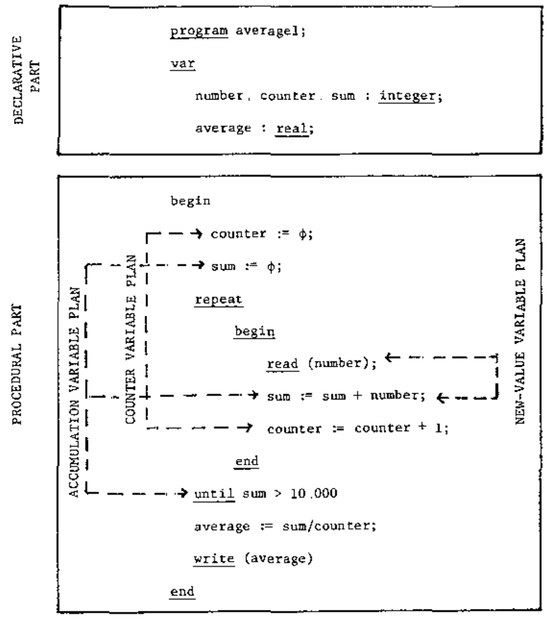
```
sum := ϕ
count := ϕ
    repeat
       read (number);
       sum := sum + number
       count := count + 1
    until count = 40

{repeat-loop solution}
```

```
sum := ϕ
for i := 1 to 40 do
     read (number
     sum := sum + number



{for-loop solution}
```

Figure 4

The fourth problem (see Figure 4) necessitates the construction of the three variable-plans associated with the variables *number, sum* and *counter*. It is important to notice that the problem contains two steps. Even if the student could solve the problem for 5 integers by using 5 different variable names (i.e., associate the name of a variable with each number), s/he has to change his/her procedure when solving the problem with 40 integers. In this latter case it is difficult to work with 40 different variable names! So s/he has to insert a read-statement into the loop.

```
P5 - Write a program which repeatedly reads in integers until their sum
     is greater than 10 000, prints out their average and the number of
     integers read in
```



```
program average1;
var
   number, counter, sum : integer;
   average : real;
```

```
begin
  counter := φ;
  sum := φ;
  repeat
    begin
      read (number);
      sum := sum + number;
      counter := counter + 1;
    end
  until sum > 10.000
  average := sum/counter;
  write (average)
end
```

Figure 5

The last problem involves the construction of three variable-plans as shown in Figure 5. Note that this problem and the preceding problem have the same loop invariant. But they differ with respect to the status of the variable on which the terminating condition operates. In problem 4 the number of repetitions is known in advance; it is equal to 40. Whereas in problem 5, the number of repetitions is not known; the terminating condition operates on the calculated value designated by the variable *sum*.

5. Conditions of the didactic experiment

The problems are given to the students during the first 10 sessions of the didactic sequence. It is important to point out that our aim is not to evaluate the students' performances. We consider the problem-solving situations function not only as a criterion of what is learned, but also as a situation in which the students encounter some new concepts. So each of the given problems constitutes for us a didactical situation in which we have introduced some new concept.

The first two problems are used to introduce the concept of repetition and the construction of the loop-invariant. The third problem is conceived to confront the student with the concept of sequentiality. In the fourth and fifth problem the students encounter for the first time a problem in which they have to use the read-statement in the loop. The *for*-loop strategy is then introduced as a most appropriate loop-strategy for problem 4. Comparison of these two problems allows us to discuss in class the idea of an appropriate looping-construct for a given problem. (Note that one cannot solve the problem 5 by using the *for*-loop.)

Three groups of 18 college students (15-16 years old) were involved in the study. Two of the groups belonged to the same class. The third group was made up from volunteers at the same college. The students worked in groups (3 per group) one hour a week during 30 weeks. Two classrooms in the college were used for the experiment. In each session students worked in one classroom first, in a class situation (with a regular teacher), and then worked on computers only, testing the procedures they had elaborated previously. The programming was done in the language PASCAL-UCSD, on six SIL'Z micro-computers.

Four mathematics teachers and four researchers participated in the experiment. One researcher and one teacher regularly undertook the teaching with the class and the voluntary group respectively. The others observed the different groups during the sessions. The experimental sessions were all preceded by a working meeting of the participating teachers and researchers. Teachers were actively involved in the elaboration and the analysis of each session.

The full analysis of the complete sequence can be found in a recent research report [Roucher, Samurçay, 1984].

6. Analysis of strategies used by students

Theoretically when experts construct a program they generally begin with the procedural part. At the beginning of their learning experience, some of the students did not follow this plan: they started with the declarative part of the program as shown in the following example:

Student writes:

```
program clair;
var
     x, y, n
            begin
              .
              .

            end
     write
```

and tells her friend to complete the rest. This strategy corresponds to a stage in which the student has not yet represented the elements of the program, and particularly the relations between them. But as seen in the above example, the student knows that some variables should be declared, and that something has to be printed out at the end of the program.

This kind of strategy disappears quickly with experience.

The second category of strategies is characterized by work on the procedural part first. We have identified four levels of strategies of this kind.

A. Procedure focusing on the result corresponding to the specific data, lack of use of variables.

This first strategy is characterized by the student's difficulty in representing the data of the problem by using the names of variables. We have also observed this kind of behaviour with 9-year-old children in LOGO sessions [Hillel, Samurçay, 1985]. For instance, while the children are able to write a procedure for a particular shape of a particular size, they cannot easily transform this procedure into a new one which generates a set of congruent shapes. Or, they may be able to use additivity when they work on particular values.

e.g. $\left.\begin{array}{l}\text{FD } 30\\ \text{FD } 30\end{array}\right\} \Rightarrow \text{FD } 60$

but don't see the same property in

$\left.\begin{array}{l}\text{FD : LENGTH}\\ \text{FD : LENGTH}\end{array}\right\} \Rightarrow \text{FD } 2^{*}\text{:LENGTH}$

Example: Solving problem 3

S1. *"We have to take 1, 3, 5, 5, 7... These numbers are prime numbers, and then add* n *of them up."*
S2. *"We have to give a value to* n, *the machine doesn't know what* n *is."*
S1 *"*n *may be anything [any number]. It should work for all numbers."*
S2. *"OK! but if you say* n *it means nothing. It can't know the number you are thinking about."*
S1 (writes: 1 + 2 + 3 + 4 + 5 + ...]
"Add 2 to 1, 3 to 2, 4 to 3, ..."
S2. (writes: = 15)
"Now it can understand."
S1 *"OK, but we don't know if* n *is 15. I think we can do it like this."*

$\underbrace{1+2}_{3}+3+4+5+6$

"Add the first two numbers, take the two others, add again and . and at the end make the whole sum."
She writes $\underbrace{\underbrace{a+b}+c}+d+e$

S2. *"Yes, but it will not work, one cannot know what the numbers* a, b, c, d *are."*

Note that for student 2, the particular values become unknown when they are associated to the name of a variable: a signifier. At this stage students can identify and construct procedures with particular values more easily than with the signifiers which name the variables. We can say that the initial conception of variable is that of an undetermined quantity.

B. Production of an algebraic form which is not procedural and not directly programmable.

Example: Solving problem 5
$a + b + c + d + \ldots = \text{sum}$
"There are 40 like that and we obtain the sum."

Example 2. Solving problem 3.
$k + (k - 1) + \ldots + 1 = \text{sum}$

We see in these two examples that students produce an algebraic rewriting of the problem. It seems very hard for students to transform these writings to procedural ones. The teacher intervened here to get students to be explicit about what they were trying to do.

C. Explicit procedures are produced, but with "collisions" between algebraic expressions, natural language formulations, and programming code elements.

We first remark that in this type of strategy there is an identification of necessary actions and also some structuration of them. But the elementary actions are not made sufficiently explicit and there is confusion about the identity of the executor of the actions: I myself or the machine?

Example: Solving problem 3.
We present here the successive writings of the same group working on problem 3. Note that the first two writings correspond to the former categories A and B that we have identified above.

1. [*writing*]
$1 + 2 + 3 + 4 + 5$
$1 + (x + 1) + (x + 1) + \ldots$
$x = 1 \quad x = x + 1$

2. [*writing*]
one prints 1
one gives to x the value 1 for the first operation
one calculates $x + 1$
for the second operation
x takes the value $x + 1$

3. [*writing*]
$(1 + (1+1) + (2+1) + (3+1) + (4+1) + (5+1) + \ldots$

We see that the major difficulty with problem 3 is the construction of the accumulation-variable which calculates successively the partial sums. The only invariant identified by the students is the law of succession of the values of the variable which runs through interval [1, 2, ..., n]. The other variable, the partial sum, is seen as a whole without decomposition.

D. Elaboration and expression of procedures involving an analysis and use of the signs of elementary actions with structuring elements

The strategies used in this category are characterized by the search for the invariant property of the actions in the loop. We distinguish two subcategories.

D1. Actions are analyzed in terms of programming operations. Obstacles with variables.

Example: Solving problem 1.

1. [*writing*]
```
(x + y) → a
b = 1
repeat
a*a
add 1 to b
until b = n
```
"multiply a *by itself the number of times indicated by* n*".*

They translate this procedure into Pascal:

```
"a := x + y
b := 1
repeat
a*a
b := b + 1
until b = n."
```

We see in this example that the counter-variable plan is well-constructed but the variable which would allow the final result to be calculated is not yet identified. The accumulation-variable plan is not constructed.

The first remark to make is that the successive values obtained by the multiplications by a are not identified. The student knows the definition of a^n but he is not able to transform this definition into a plan of calculation. The knowledge of the successive values of a, a^2, a^3, ... is not sufficient to construct the invariant relation between the different elements of the loop.

The second remark is that the counter-variable plan is constructed independently of the accumulation-variable plan. One may suppose that there is not an achieved conceptual acquisition here, but rather an acquisition of a form: "the counter must be initialized to zero, then incremented by 1".

D2. Actions are analyzed and there is representation of the specific values of the variable on which they operate.

Example: Solving problem 5

1. [*writing*]

```
repeat
read the number and add it to the sum
a := a + x (the number)
until a > 10,000
and b = b + 1
type +1 for each number added
```

2. [*writing*]

```
S1   read (x)
     repeat
     a := a + x
     b := b + 1
     until a > 10,000
```

S2. *"You have to put* "read" *in the loop!"*

S1. *"No, because you can only put the orders in a loop".*

Ob. *"What is* "read"*?"*

S1. *"It is some information that we are giving to the machine".*

3. [*writing*]

Ob. *"What is the initial value of a* in a loop?"

S2. *"It's zero, at the beginning there is nothing in it".* (He modifies the program)

```
Repeat
a := 0 + x
b := b + 1
until a > 10,000
```

This example shows quite clearly that, even where the students are able to construct the accumulation-variables correctly, they have some trouble with the meaning of the read command and the initialization. Concerning the read command, the example shows that the read operation is first treated correctly in the elaboration phase of the loop (i.e. it is inside the loop), but the problem of meaning is encountered in the expression of the program in PASCAL. We see in this behaviour a possible confusion between the actions of the automaton and the voluntary actions of the subject. In fact when the read command is used, the subject "gives some information (data)", but the machine needs a specific command to receive it.

Concerning the initialization problem, we remark a similar phenomenon: the loop-invariant is well-constructed at first, but the intervention of the experimenter is sufficient to show the instability of the student's conception.

The initialization of the variables is the major difficulty encountered by beginning students. We observed that the obstacles relating to the initialization problem did not disappear at all quickly during the learning sequence. This problem is closely related to the conceptualization of the notion of variable itself [Samurçay, 1984].

7. Concluding remarks

In this paper we have presented a description and an analysis of general strategies used by students in problem solving involving loops. We have categorized four types of hierarchized strategy. Each of these strategies is characterized by a certain level of conceptualization of the notion of variable and the construction of loop-invariant.

Our findings show that at the end of the 10 sessions of programming, the students have serious troubles with the transformation of their algebraic description of the given problem into a procedural description. We argue that the algebraic conceptions of variable, equality sign, and equations, constitute a necessary but an insufficient model on which to build the programming concepts of variable, assignment and loop-construct.

We have noticed that initialization is a very difficult operation, even for students who realize a high-level strategy in the construction of a loop invariant. We think that the initialization problem is not specific to the domain of programming in Pascal. Some of the errors we have observed in LOGO programming [Hillel, Samurçay, 1985]—particularly in interface problems—can be analyzed to show the same obstacle: how to make an hypothesis about the initial state, knowing the final state and the transformation. For example, when children have to elaborate an interface between two procedures P_1 and P_2, their problem can be formulated as follows:

$$\text{final state of the turtle in } P_1 \xrightarrow{\text{interface}} \text{initial state of the turtle in } P_2$$

We have observed that children often have a lot of difficulty in constructing the interface procedure, which is in fact the initialization part of the second procedure.

The complexity of this last problem, with additive structures, has been explored by many authors [e.g. Carpenter, Moser, Romberg, 1981]. The difficulties with the concept of variable are more durable; a more specific study on variable is suggested by our recent work [Samurçay, 1984].

Finally, we allow that the stages analyzed above are not definitive, but they reflect a first attempt to represent the evolution of student's programming procedures.

Acknowledgment

This work was done in collaboration with A. Rouchier, J Rogalski, C Landre, J.C. Despland, J.M. Laubin, J.F Pigeonnat, Y Ferrand, G Sarfati, G Vergnaud.

I would like to thank D. Wheeler for his many helpful comments

References

Bonar, J., (1984) *Understanding the bugs of novice programmers* Technical Report, LRDC, University of Pittsburg

Carpenter, T P., Moser, J.M., Romberg, T.A., (eds.), (1981) *Addition and subtraction: a cognitive perspective.* Hillsdale: Lawrence Erlbaum

Hillel, J, Samurçay, R., (1985) *Analysis of a LOGO environment for learning the concept of procedures with variables* (in preparation)

Pea, R.D., (1984) *Prospects and challenges for using microcomputers in school.* Technical Report No. 7. New York: Center for Children and Technology, Bank Street College

Pea, R D., Kurland, D M, (1983) *On the cognitive prerequisites of learning computer programming,* Technical Report No. 18 New York: Center for Children and Technology, Bank Street College.

Pea, R D., Kurland, D.M., (1984) *On the cognitive effects of learning computer programming: a critical look,* Technical Report No. 9. New York: Center for Children and Technology, Bank Street College.

Rouchier, A., Samurçay, R., Rogalski, J., Vergnaud, G., Despland, J C, Landre, C., Laubin, J.M, Sarfati, G, Pigeonnat, J.F, Ferrand, Y, (1984) *Concepts informatiques et programmation Une première analyse en classe de seconde des lycées,* Rapport de recherche. CNRS-Université d'Orléans

Samurçay, R., (1984) Signification et fonctionnement du concept de variable informatique chez des élèves débutants Submitted to *Educational Studies in Mathematics*

Samuçay, R., Rouchier, A, De "faire" à "faire-faire": planification de l'action dans une situation de programmation To appear in *Enfance,* 1984

Soloway, E., Ehrlich, K., Bonar, J, Greenspan, J., (1982) What do novices know about programming, in *Directions in Human-Computer Interactions,* B Shneiderman and A Badre (eds.), Ablex Publishing Co.

Vergnaud, G., (1982) Cognitive and developmental psychology and research in mathematics education: some theoretical and methodological issues, *For the Learning of Mathematics,* 3, 2: 31-41

Continued from page 36

the various groups, as did the degree of success in catering for teachers' interests. There was indeed, a high proportion of teachers among the participants and their responses seemed generally favourable. The Congress provided them with the opportunity to hear and to talk with people of international repute; they were able to exchange ideas and perceive shared problems with confrères from many parts of the world; and it was an occasion which enabled them to measure the degree to which their own thinking matched levels elsewhere. We are still too close to ICME-5 but it could be that from the events in Adelaide a great deal of productive, co-operative achievement might flow. The potential for this will depend on the ability of the groups' chief coordinators to sustain the impetus. It is too easy for established routines and immediate local pressures to take precedence over new, long range and remote initiatives. Topics which tended to recur in several sections during ICME-5 were: the role of problem-solving and modelling; the need to translate research findings into effective classroom practice; the high priority that should be given to pre-service and in-service teacher education; the important place of language in mathematics; and the need for mathematics learning to be a socially interactive process. A useful and much appreciated feature of the Adelaide program was the provision of periodic summaries of proceedings of certain groups. This was one way of resolving the difficulty of conflicts in interests and timetabling

Finally, one of the pervading influences of ICME-5 was the Cockcroft Report. This seems to have risen like a beacon, providing direction and signalling support for many positive issues that have emerged in mathematics education over the past 25 years: active learning, social interaction, mathematics as a mental process, flexibility in one's repertoire of teaching skills to match the varying needs of learners and so on.

Out of all this, what does ICME achieve and what is its future? Mathematics education has witnessed its own knowledge explosion over the past 25 years. Consider the three-volume research report that was fed to the Cockcroft Committee. ICME has a continuing and important role to play in bringing people in related fields together for the sharing of information and ideas. The future strength of ICME will depend, it seems to me, however, on its ability to preserve the balance between information-giving and the involvement of participants. One way to achieve this is through the continuity which Working Groups might provide in the form of reports on the present state of affairs, new issues, problems, solutions and practices and the like. Such working groups should be allowed to rise and fall as the need demands. It would seem important also to maintain the Action Groups, from early childhood to tertiary levels, and to preserve another balance: that between mathematics and mathematics education. Finally, the difficulty of finding satisfactory solutions to the associated problems of publicising research and improving practice would seem to mandate in favour of making every ICME equally accessible and attractive to the mathematics educator/researcher and the mathematics teacher at every level.

JOHN S. CONROY
Faculty of Education
Macquarie University
Sydney, N.S.W. 2113
Australia