Problem decomposition using programming plans

Y.A.J. Berendsen, H.P.M. Krammer
Toegepaste Onderwijskunde
Universiteit Twente

Abstract

The use of programming plans is an important method for decomposing problems. The goal of this study was to gain insight into the usefulness of programming plans in teaching introductory computer programming. Subjects were students at a school for higher vocational education. Subjects' use of programming plans was observed by the inspection of programming products and the analysis of thinking-aloud protocols. The results indicated that the concept of programming plans given to the subjects must be told explicitly in order to get them to use these plans.

1. Introduction

Courses in computer programming are an essential part of the curriculum of technical vocational education. Some introductory skills in programming are considered essential, not only in training courses for computer engineers but also for other technical occupations such as chemical analyst, architect and engineer.

In introductory programming courses structured programming is generally emphasized. The reasons for this are not always clear. First, in software development a structured approach is necessary (Dahl, Dijkstra, & Hoare, 1972; Wirth, 1974; Yourdon, 1975). For several technical occupations, on the other hand, programming is not supposed to be a necessary skill, since relevant applications are available for most tasks. Second, a structured approach may be taught in introductory programming courses because of the expected effects on general problem solving abilities. However, research presents conflicting results (Rucinski, 1991). Whatever the reasons, teachers and textbook authors apparently think it necessary to teach a structured approach in introductory programming courses.

Generally, the instructions for a structured approach to computing comprise expository statements, the presentation of worked examples and a design scheme. The latter appears in different forms: flow-charts, structured charts, Nassi-Shneiderman charts, etc. (Brooke, & Duncan, 1980; Green, 1982; Shneiderman, Mayer, McKay, & Heller, 1977). However, most students of introductory computer programming courses have problems in applying these

techniques. Usually they do not use the design schemes, since they want to start the coding of the variables and the operations to solve the problem as soon as possible. They do not pay attention or may be unable to design the general structure of the program before coding the variables and operations. Obviously for the students programming means coding. Tromp (1989) phrases the motto of such students as: 'Code first, repair later and design never' (p. 47). This idea may be strengthened because the usefulness of the design techniques which are presented to students are unclear. Often the techniques are too detailed, too complex or too time consuming to use. Secondly, the instructions for the use of design techniques may be inadequate. It is assumed that the content of the instruction emphasizes the syntax and semantics of the programming language. As soon as the teacher finishes the expository part of the instruction, the students are haphazardly proceeding with their work. Results of a previous experiment showed that in spite of the presented design techniques, the teachers of introductory computer programming courses did not provide feedback on the students' design, or even worse did not see the designs. The authors suppose that when the students are developing the program, the provision of feedback and explicit instruction of the design scheme is essential.

One of the key steps in designing schemes is the analysis of the problem and the synthesis of a set of subproblems that solve the original problem. A generally used term for this proces is *decomposition* of the problem (Jeffries, Turner, Polson & Atwood, 1981). The fact that decomposition plays a minor part in the programming activities of the novices, implies that the students should be coached during their problem solving. The usefulness of the design scheme can be promoted if the student learns and practices the decomposition skills. They should be coached during their decomposition trials. This implies that the teacher gives informative feedback on intermediate and final programming products. In doing so, the student receives relevant comments on his decomposition actions and possible frustrations of the student might be overcome.

This paper addresses the question of how the decomposition can be represented in instruction in order to further the acquisition and to make instructional feedback possible.

2. Decomposition and programming plans

The term decomposition refers to two different concepts. First, decomposition is described as the *process* of decomposing the problem into more manageable units. Second, the term can refer to the *product* of this process. The latter can be rather abstract (e.g., if it only resides in the brain of the programmer) or more concrete (e.g., in the form of a diagram). Our observations illustrate

the difference between these two meanings. Some students who had to solve a programming problem proceeded in a rather unstructured way. They 'hacked' (programmed quick and dirty) and debugged as they used to do when they still made BASIC programs on their home computers, but they took a final step to their solution by reorganizing the program into procedures and sub-procedures with the addition of comments, so that the programs looked like they were developed in accordance with the principles of structured design. This observation illustrates that a proper decomposition *product* (as reflected by a program) may be developed in accordance with an undesirable decomposition *process*.

A main goal of most programming courses in vocational education is that students acquire the process of decomposing problems based on structured design principles. In order to reach this goal, the instruction should contain a suitable model for the product of a decomposition. Possibly the teacher can use this model to demonstrate the process, and to give feedback on students' decompositions.

For practical reasons, the model for a decomposition product should meet the following requirements. First, the visual reproduction of the decomposition should be chosen in such a way as to clearly represent how to approach the problem, both for the teacher and the student. Second, the usage of the decomposition product must be attractive to the students. This means that the student should not spend too much of his/her time and effort making a representation of the problem approach. Furthermore, the instruction for making a decomposition product has to be as simple as possible and even has to be linked up with the current programming curriculum. Fourth, the approach to the problem and the structure of the program must be deducible from the decomposition product. Finally, the decomposition as an intermediate product should simplify the student's coding task.

A substantial number of empirical studies support the assumption that experts in domains like chess, go, geometry and electronic circuits, use goals and plans in their problem solving strategies (Chase & Simon, 1973; Greeno, 1978; Soloway, 1986). Studies of experts' programming clearly demonstrate the use of programming plans (Adelson, 1981; Johnson, 1985; Rich, 1981; Rist, 1986; Soloway, 1985, 1986; Spohrer, Soloway & Pope, 1985). These programming plans are standard fragments of code to achieve certain goals. For every subproblem (which can be considered as a specific goal), several standard solutions (which can be considered as plans) are available. The plans consist of lines of code which belong together to achieve a particular goal.

The term *plan* may suggest that experts always consciously design their programs which is not necessarily the case. Within the framework of programming other terms are in use, namely schemes (Gegg-Harrison, 1991),

templates (Linn & Dalbey, 1985), beacons (Wiedenbeck, 1986), or cliches (Vanneste, Olivié & Dedecker, 1990). Because of its widespread use we hold on to the term *plan*.

A programming plan does not necessarily form an integrated part of the code. It is in fact a framework with its 'feet' (which are represented by the lines of code) on different locations within the program. Figure 1 shows three programming plans: 'repeat-until', 'running-total', and 'count-how-many'. The running-total plan, for example, keeps the total of the numbers read or, as Figure 1 makes clear, keeps the total of the counts. This plan consists of two subplans, one which sets the total at zero and the second which increases the total within the loop. Figure 1 further shows that the running-total plan covers two different lines of code, which are positioned at two different locations in the program. However, these lines belong together in order to keep a running total. The figure also demonstrates that in the final program several plans are intertwined with each other. In order to get a working program, the programmer has to put the plans together in a good way. This composition of plans is a particular source of many novices' misconceptions (Spohrer & Soloway, 1986).

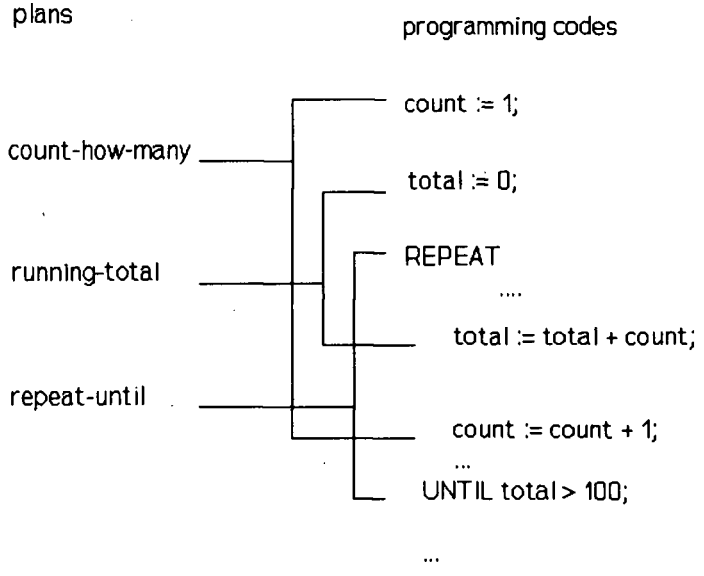


Fig.1. Illustration of the relation between plans and code for three plans and the corresponding Pascal program

Experts distinguish several plans in their programs, whereas novices usually only see the individual code lines of the program. A major feature of programming expertise is the use of these plans. Experts have built a whole repertoire of such plans which is used as a kind of collection of ready-made answers to problems (Soloway, 1986).

According to Spohrer, Soloway and Pope (1985), several misconceptions of novices can be traced back to the difference between psychological plans and programming plans. Psychological plans are used to achieve general goals from the 'outer world', whereas programming plans are used for achieving goals from the 'programming world'. Novices have not yet developed several plans to handle their problem, so they are looking for the solution with little planning and organisation (Rist, 1989; Soloway, 1986). Various researchers recommend providing the novice with explicit instructions in relation to programming goals and plans (Bonar, Riggs, Weil & Jones, 1987; Rist, 1989; Soloway, Ehrlich, Bonar & Greenspan, 1982; Spohrer & Soloway, 1986).

Taking into account the assumption that experts make use of goals and plans in their problem solving strategies and the requirements which are described above, the authors developed a model for the decomposition product. In presenting this model to novices, we wanted to know to what extent the subjects did actually use the plans in decomposing the problems so that the model's usefulness might become clear. Therefore we wanted to know something about the subjects' cognitive processes.

3. Method

Subjects

Twenty-one subjects (15 male and 6 female, about 18 years old) who were requested to volunteer, participated in this study. The subjects were chemical technology freshmen at the Hogeschool Enschede, a school for higher vocational education. They attended classes of introductory programming (Pascal) and they had followed a maximum of five lessons in introductory programming. During four lesson periods, the subjects participated in the experimental treatment which was a part of the normal school curriculum. The subjects were not paid for their participation.

In addition, verbal protocols of thinking aloud procedures were collected from four subjects (1 female and 3 male). These procedures took place outside of the regular lessons. These subjects were paid for their cooperation.

Materials

The instructional material (note 1) described several programming plans, when to select and how to compose them. The subjects were to use the

instructional material together with a textbook which discussed how to use Nassi-Shneiderman Diagrams (NSD's) for the representation of the three most important programming structures, viz. sequences, iteration and selection. In the textbook several problems of increasing difficulty were presented which had to be solved with the help of NSD's. The problems presented in the textbook were also used for practising the programming plans.

Test

A short questionnaire (nine items) was administered to estimate the subjects' programming experience. Questions like "have you ever written a program?" or "have you ever read a program, that is to say did you understand the lines?" had to be answered by the subjects.

Verbal protocols

Finally, three problems were used to collect thinking aloud protocols which were audio-taped.

Procedure

During the first lesson subjects were asked to fill in the short questionnaire on programming experience.

After most of the subjects had studied the instructional material, they were asked to select several problems from the textbook and to produce a decomposition based upon the presented plans. Since the subjects were novices their text-book consisted of, among other things, simple programming problems. In order to make clear the concept of a plan, it seemed useful to practise these plans on very simple problems. Beside this, the plans are in this way linked up with the current programming curriculum.

The decomposition based upon plans had to be worked out as follows. They first had to select the appropriate plans, subsequently they had to compose these plans and finally they had to make an NSD of their composition. With this last step the subjects could participate with minimal effort because they had to develop an NSD anyway (it was in the school's curriculum). They were allowed to solve the problems during class or at home. The decompositions had to be handed in to the researcher.

After all the decompositions were handed in, an appointment was made with the four subjects for recording the thinking-aloud protocols. These subjects were first given an example problem in order to get them used to the task of thinking aloud while solving a problem. After they had solved the example problem, they were given a possible solution and received oral feedback on their own solution. The remaining three problems were presented

without discussing their solutions. Every subject required about fifty minutes to solve all the problems, except for one subject who could only solve two problems.

Analysis

The decomposition product and the verbal protocols constituted the data to be analysed. In both analyses, the main question is whether the subject actually used the plan-based decomposition method. Therefore, the following issues were of importance. First, did the subject make a selection of appropriate plans? Second, did the subject compose these plans in a correct way? Third, are subplans, if any, being noticed? Fourth, did the procedure of decomposing proceed from plan selection to NSD specification?

Analysis of the product - The decomposition notes were evaluated with the help of a worked-out scheme with plans. In these worked-out schemes the correct plans are selected and composed in accordance with our method (see Figure 2 where one of the problems is worked out). Because the subjects

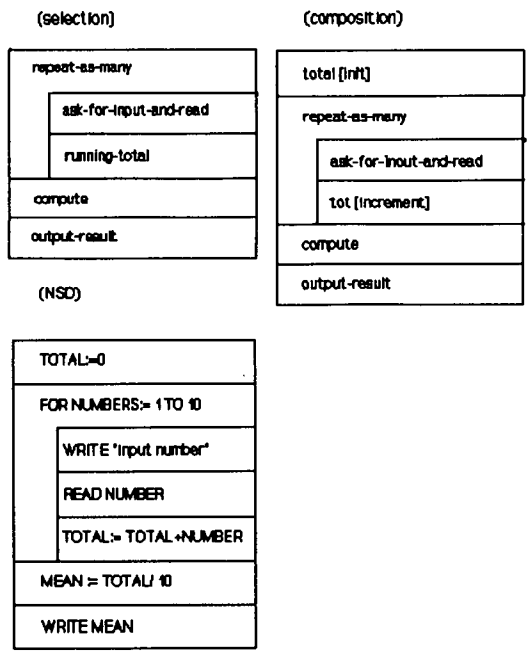


Fig.2. Worked-out scheme, according to our decomposition method, of the following problem: Read in ten numbers and compute the average of these numbers and assign this to the variable MEAN. Print the output of MEAN.

were all novices, an incorrect or partly correct use of plans was expected. Therefore, we wanted to check whether (a) a minimum of three plans was correctly selected, (b) a minimum of three plans was correctly composed, and (c) the course of the programming process started with plan selection and ended with NSD specification. We checked for each subject whether the minimum of three plans was correctly *selected*. If this was true the subject scored one point otherwise they scored no points. The same procedure was used for the *composition* of a minimum of three plans and for the *procedural sequence* of the subject. In relation to the latter evaluation it must be noted that we checked whether the subject had begun with plans: if the subject had only made an NSD, a zero score was assigned because he/she had immediately started with NSD, if he/she had first noted plans one point was assigned. Thus the most important issue here was whether the subject had begun his decomposition product with plans. We expected to find at least 70% of the subjects had met the requirements we just mentioned.

Analysis of thinking aloud protocols - Thinking aloud sessions were audio recorded and typed out. The protocols of all three problems were scored on the occurrence of verbal references to plan-structures and plans. Two questions with regard to the scoring of these thinking aloud protocols were: (a) Did the subjects make use of plan-structures and plans? (b) Did they proceed from plan selection and plan composition to NSD or code specification? We conceived the cognitive activity of plan selection and plan decomposition as high, and NSD or code specification as low abstraction. When the subject had indeed used the plans but applied them retrospectively, that is after the coding, the subject didn't proceed in accordance with the authors' idea of decomposition.

The following scheme was used to observe subjects' application of plans in which the subjects' formulations which referred to plan-like structures were coded. Five categories were distinguished, four of which were interpreted as levels of abstraction, the fifth was used for cases that were difficult to categorize.

Level 1 - The use of the *literal plans*, i.e. the plans which exactly match the terminology as described in the instructional material. For example, 'ask-for-input-and-read plan', 'compute plan' and 'test plan'.

Level 2 - The use of the *paraphrased plans*, i.e. plans which do agree with the plans from the instructional material, but are named in the subject's own words. For example, 'that has to be printed' which is in agreement with the plan 'output-plan'.

Level 3 - The use of sub-plans. These may be the sub-plans as formulated in the instructional material or may be self-labeled as well. For instance, 'count

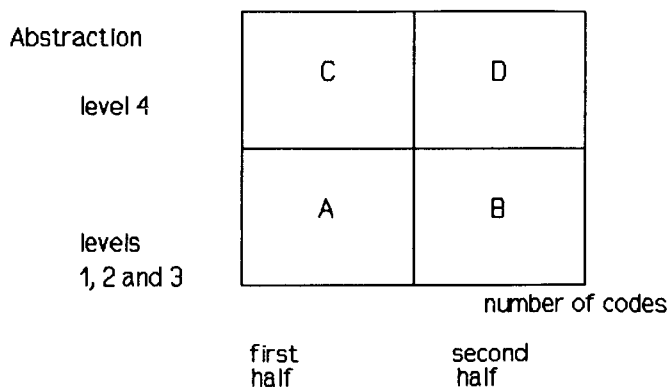
[initialize]' (literally as described in the instruction) or 'the counter is set to zero' (paraphrased)'.

Level 4 - The use of code or NSD terms. Examples are, 'for number is 1 to 10' or 'while .. do'.

When cases were difficult to categorize, they had to be included in the remaining category. An example is 'write x'. In this case it is unclear whether the subject is paraphrasing 'output-plan' or referring to code 'WRIT-E(x)'.

According to the required procedure it was to be expected that the subjects should mainly report plans in the first half of the thinking-aloud protocols, while in the second half the more detailed terms should be predominantly present.

For the analysis of the text of the thinking aloud protocols, an index of *planning sequence* was constructed. The first three levels were taken together as being a level of high abstraction opposite to level 4, a level of low abstraction. For each problem's protocol a distinction was made between the first half and the second half by taking the median of the coded scores of every part of the protocol (each protocol consisted of three parts, namely the three problems). The index of planning sequence was computed as being the fraction of codes that were in order, that is to say, firstly the codes of the level(s) of high abstraction and finally the codes of low abstraction. Figure 3 illustrates this definition.



Note: The numbers a, b, c, and d are cell frequencies. The index of planning order is defined as (a + d)/(a + b + c + d)

Fig.3. Partitioning of the code frequencies according to abstraction level and number of codes

The index of planning sequence was expected to be greater then .70. If this were the case we might conclude that the requested procedure did actually take place. Subsequently we could evaluate if it would be useful to continue with this procedure in a follow-up study.

4. Results

The products of programming

In total 29 decompositions made by 14 subjects were taken in. Most of the subjects handed in two or three products. After a first review of these products, we decided to take the subject's best decomposition into the analysis. The best decomposition was the one which most agreed with the authors' idea of decomposition.

From the analysis of these decompositions, it appeared that eleven out of fourteen subjects (79%) made a correct selection of a minimum of three plans. However, 43% of the subjects made a correct composition of a minimum of three plans. A correct sequence of the decomposing process - first focusing on plans and eventually ending in NSD's - was found for 86% of the subjects.

Thinking-aloud protocols

Two independent observers scored the protocols, with an inter-observer reliability of .80. In Table 1 the index of planning sequence is presented for all subjects and all programming problems. It appears that the average index per person of all the three problems together, ranges between .58 and .82. We find six out of the eleven segments of protocol to have an index greater then .70. That means that the requested method can be recognized in the majority of the protocol segments (55%).

Table 1: Index of planning sequence for four subjects and three problems

	Problem			
Subject	1	2	3	Mean
1	.78	.47	.50	.58
2	.50	.67	.60	.59
3	.77	.86		.82
4	.81	.78	.72	.77

Two typical examples of parts of thinking aloud protocols, collected from two subjects, follow below. The subjects tried to decompose the following problem: "the numbers from 1 up to and including 100 must be added and the result must be printed".

> "Iteration plan, a computing plan...and after this all a printing plan...thus A becomes...until the counter at 100 so...add up the total...and print the result".
> (Subject nr. 2)

This example shows that the subject immediately labeled the plans. According to him he needs an iteration plan, a computing plan (which is actually not correct because he should have mentioned a running-total plan) and a printing plan. No details are given. It certainly is clear that this subject has not yet worried about the details, which is in our opinion an encouraging procedure.

> "...Begin with initializing a counter...total is zero...he has to add afterwards eh T is T plus one...and he does this every time...and eh then
> ... the counter is, he starts with, he sets it also to zero...then of course we are going to do the same build-up just like before...and we know what he is doing here below...we put above it 'FOR 1 TO 100', total is zero and we set the total to T and he adds it every time...difficult..".
> (Subject nr.1)

This subject typically has directed his attention to the details and he is therefore not proceeding in accordance with our method.

Questionnaire

The short questionnaire on programming experience which the subjects had to complete showed that everyone had seen a program at least once, 76% did understand the lines of the program and 71% had written a program once before.

5. Discussion

Both the subjects' plan selection (79% of the subjects), and their procedures of working from an abstract level to a concrete level (86% of the subjects) satisfied the preset criterion of 70%. However, the subjects did not meet the criterion for the composition of the plans (43% of the subjects).

The subjects are left free in the selection of a programming problem from their textbook. This probably implies that they do not select the most difficult problem(s). Some of the problems are easy to solve so that a composition of the plans with an indication of the subplans, is not necessary because the composition appears in the NSD. The easiness of the problems thus may explain the low percentage of plan-composers (43%). Moreover as was stated

earlier, during the introductory programming classes the NSD's were being practised. So it is possible that the NSD's may have acquired a goal function which directs the subject's attention to the eventual NSD at the cost of the composition of the plans.

One of the misconceptions occurring most strikingly in these products, is related to the idea of a plan. Four subjects did not make use of the described plans in their decomposition, but had described their contemplation. So the 'programming plans' are sometimes considered 'psychological plans' by the subjects. Three times, plans were used in an ambiguous way. For instance, one subject initially selected the plan 'repeat-n-times', but in his composition this plan had suddenly disappeared and was exchanged for the plan 'repeat-until'. Four of the subjects used 'running-total' or 'count-how-many' plans to solve a problem, but did not mention these plans in their selection. However, they did point out the sub-plans (for example: count [init] and count [increment]) in their composition. It is most likely that these subjects did not understand that the sub-plans they mentioned in their composition actually belong to the 'running-total' and 'count-how-many' plans. One subject wrote beneath his decomposition product that he proceeded the other way round; he first made a NSD and then derived a decomposition, because the problems were too easy to do in the way we asked him. Finally, subjects often used the concepts 'compute' instead of 'keep a running total'. Probably they did not understand the idea of the 'running-total' plan which is always present within a loop.

The analysis of the thinking aloud protocols reveal that 55% of the protocol segments have an index greater than .70, which is satisfying enough to proceed with our method in a follow-up study. The authors cautiously conclude that most of the protocol segments have reported the requested thinking procedure. They proceed from the highest level of abstract to the lowest level.

With regard to the results of the questionnaire we might conclude that at least some of the subjects had more experience than was expected.

The results of both the products and the protocols support the conclusion that the majority of the subjects do apply the required method in a correct way. They proceed from an abstract level to a concrete level by starting with plans and ending with a NSD. In further research the authors will continue with the use of the method. The instructional material will be revised by taking into account the remarks made by the subjects. For instance, the differences between all the plans should be explained more explicitly. In addition, more examples must be discussed. The effectiveness of the plans will be studied. Thereby we will give feedback on the decompositions, in order to emphasize the design aspect of programming. By explicitly offering

programming plans to the students, and subsequently giving feedback on their decompositions, it is expected that the beginning programmers will learn to program more effectively and with greater ease.

Note:

1. "Plannen en PSD's: een experimentele uitgave in het kader van het TUTOR-project, januari 1990" [plans and PSD's: an experimental publication within the scope of the TUTOR-project, january 1990] by Yolande A. J. Berendsen, Department of Instructional Technology, University of Twente.

Literature

Adelson, B. (1981). Problem solving and the development of abstract categories in programming languages. *Memory and cognition, 9*, 422-433.

Bonar, J., P.Riggs, W.Weil & R.Jones (1987). *Programming plans workbook.* Experimental edition.

Brooke, J.B. & K.D.Duncan (1980). Experimental studies of flowchart use at different stages of program debugging. *Ergonomics, 23*, 1057-1091.

Chase, W.C. & H.A.Simon (1973). Perception in chess. *Cognitive Psychology, 4*, 55-81.

Dahl, O.J., E.W.Dijkstra & C.A.R.Hoare (1972). *Structured programming.* London: Academic Press.

Gegg-Harrison, T.S. (1991). Learning Prolog in a schema-based environment. *Instructional Science, 20*, 173-192.

Green, T.R.G. (1982). Pictures of programs and other processes, or how to do things with lines. *Behaviour and Information Technology, 1*, 3-36.

Greeno, J.G. (1978). A study of problem solving. In R. Glaser (Ed.), *Advances in instructional psychology.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Johnson, W.L. (1985). *Intention-based diagnosis of errors in novice programmers.* (Tech. Rep. No. 246), Ph.D. Thesis New Haven: Yale University, Department of Computer Science.

Jeffries, R., A.A.Turner, P.G.Polson & M.E.Atwood (1981). The processes involved in designing software. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition* (pp. 255-283). Hillsdale, NJ: Lawrence Erlbaum Associates.

Linn, M. & J.Dalbey (1985). Cognitive consequences of programming instruction: Instruction, access and ability. *Educational Psychologist, 20*, 191-206.

Rich, C. (1981). A formal representation for plans in the programmer's apprentice. *Proceedings of the 7th International Joint Conference on Artificial Intelligence* (pp. 1044-1052), August 1981.

Rist, R.S. (1986). Plans in programming: Definition, demonstration and development. In: E.Soloway & S.Iyengar (Eds.), *Empirical studies of programmers*. Norwood, N.J: Ablex.

Rist, R.S. (1989). Schema creation in programming. *Cognitive Science, 13*, 389-414.

Rucinski, T.T. (1991). Effects of computer programming on problem solving strategies. *International Journal of Instructional Media, 18*, 4, 341-351.

Shneiderman, B., R.Mayer, D.McKay & P.Heller (1977). Experimental investigations of the utility of detailed flow-charts in programming. *Communications of the ACM, 20*, 373-381.

Soloway, E. (1985). From problems to programs via plans: the content and structure of knowledge for introductory Lisp programming. *Journal of Educational Computing Research, 1*, 2, 157-172.

Soloway, E. (1986). Learning to program = learning to construct mechanisms and explanations. *Communications of the ACM, 29*, 9, 850-858.

Soloway, E., K.Ehrlich, J.Bonar & J.Greenspan (1982). What do novices know about programming? In: A.Badre & B.Shneiderman (Eds.), *Directions in human/computer interaction*, 27-55, Ablex, Inc.

Spohrer, J.C. & E.Soloway (1986). Novice mistakes: Are the folk wisdoms correct? *Communications of the ACM, 29*, 7, 624-632.

Spohrer, J.C., E.Soloway & E.Pope (1985). *A goal/plan analysis of buggy Pascal programs*. Research Report #392. Yale University New Haven C.T, Department of Computer Science.

Tromp, Th.J.M. (1989). *The acquisition of expertise in computer programming* (Dissertation). Amsterdam: Thesis Publishers.

Vanneste, P., H.Olivié & B.Dedecker (1990). *Artificiële intelligentie en het gebruik van de computer in het onderwijs: Nieuwe perspectieven* [Artificial intelligence and the application of the computer in education: New perspectives]. Rapport ITS-1. Interdisciplinair Research Centrum, KULAK, Univ. Kortrijk.

Wiedenbeck, S. (1986). Beacons in computer program comprehension. *International Journal of Man-Machine Studies, 25*, 697-709.

Wirth, N. (1974). On the composition of well-structured programs. *Computing Surveys, 6*, 247-259.

Yourdon, E. (1975). *Techniques of program structure and design*. Englewood Cliffs, NJ: Prentice-Hall.