J. EDUCATIONAL COMPUTING RESEARCH, Vol. 6(3) 265-285, 1990

STRATEGIES FOR PROGRAMMING INSTRUCTION IN HIGH SCHOOL: PROGRAM COMPLETION VS. PROGRAM GENERATION

JEROEN J. G. VAN MERRIËNBOER
Department of Instructional Technology
University of Twente, The Netherlands

ABSTRACT

In an introductory programming course, the differential effects on learning outcomes were studied for an experimental instructional strategy that emphasized the modification and extension of existing programs (*completion* strategy) and a traditional strategy that emphasized the design and coding of new programs (*generation* strategy). Two matched groups of twenty-eight and twenty-nine high school students from grades ten through twelve volunteered for participation in a ten-lesson programming course using a small subset of the structured programming language COMAL-80. After the course, the completion group was superior to the generation group in measures concerning the construction of programs; furthermore, it was characterized by a lower mortality. The data indicated that the completion strategy facilitated the use of templates; however, this does not necessarily seem to imply that the students actually understood the working of those templates, because no differences occurred in the ability to interpret programs. In the conclusion, the completion strategy is considered to be a good alternative to more traditional strategies and recommendations are made for further improvements.

In recent years, the teaching of computer programming in high schools has become a popular activity in most western countries. However, the evidence continues to accumulate that students usually do not learn to program very well [1–7]. Whereas the research concerning the psychological processes involved in elementary programming is quickly developing, it appears difficult to apply this knowledge to the design of introductory programming courses. Consequently, there are still few guidelines or instructional strategies that teachers and others in the educational field can use to improve their programming courses in order to increase learning outcomes.

The goal of this study is to investigate if an instructional strategy which initially emphasizes the modification and extension of existing, well-designed programs (*completion* strategy) yields higher learning outcomes for a program construction task than a strategy that immediately emphasizes the design and coding of new programs (*generation* strategy). Both strategies were implemented in a ten-lesson, introductory programming course for high school students in grades ten through twelve. The two groups of students learned to use a small subset of the programming language COMAL-80.

Up to the present, no empirical data regarding learning outcomes for instructional strategies that emphasize program completion have been reported. Research concerning the teaching of elementary programming is usually aimed at instructional methods, or tactics, that are implemented using the complete generation of new computer programs. Well-known examples include the presentation of concrete computer models and the explicit teaching of design and debugging techniques [8–13]. Whereas such tactics certainly are valuable to improve programming instruction, changes on a strategic level may be equally necessary to reach higher learning outcomes.

Deimel and Moffat promoted such a change in instructional strategy by separating four phases in an "ideal" introductory programming course [14]. In the first phase, students run working programs, observe their behavior, and evaluate their strengths and weaknesses. In phase two, students are actually introduced to well-structured programs; their primary activities in this phase are reading and hand tracing of programs. Thus, learning the specific language is largely done by extracting the language features from concrete programs. During the third phase, students modify and extend existing programs and practice both design and coding aspects on a modest scale. Finally, students should be able to generate programs on their own and continue practicing basic design techniques and structured coding.

Several authors recommended similar strategies to overcome difficulties in elementary programming. Dalbey, Tourniaire, and Linn reported that students showed a serious lack of planning in program generation [15]. They suggested that "it would seem quite appropriate to begin instruction with comprehension of program code.... Those programs would demonstrate how planning is used in programming.... Thus, students would have a better understanding of the role of planning in programming" [15, p. 18]. Pea reported negative effects of "bugs" on the learning process in programming [16]. Bugs are misconceptions that students have about the operation of computers and the working of programming languages. These misconceptions cause systematic errors in the comprehension and construction of programs. According to Pea, "bugs like these could be snared if one used program *reading* or debugging activities as central components of programming instruction" [16, p. 34].

Thus, several authors have recommended that programming instruction should initially emphasize working with existing programs instead of generating new programs. With reference to those suggestions, I have investigated the psychological support that could be given to such instructional strategies, such as the reading approach, and compared them with two other groups of prevailing strategies, the expert, and the spiral approach [17]. The reading approach recommends that students begin by observing existing programs and then modifying and enhancing those programs; the expert approach stresses top-down design and starts the novice off with a relatively complex but intrinsically motivating programming problem; the spiral approach is a parallel syntax and semantic acquisition emphasizing small incremental steps and building up a program by mastering the language constructs first.

For our present purposes, the completion strategy may be seen as an instance of the reading approach, because the students' activities vary from reading and tracing, through modification and amplification, to designing and coding complete programs. The generation strategy may be seen as a variant of either the expert or the spiral approach, because it employs program generation as a primary student activity. In the following, the two main reasons for proclaiming that the reading approach is superior to the expert and the spiral approach, or more generally, that the completion strategy is superior to the generation strategy, will be briefly reviewed:

First, there must be enough task variation in practice to develop a broad procedural knowledge base, which is the basis of some flexibility in the construction of programs [18]. Such variation in elementary programming may be offered by:

1. The assignment of different tasks, such as using the editor, running and tracing programs, designing and coding algorithms, modifying, extending and debugging programs, and so forth, and
2. The presentation of a broad range of both programming problems that have different underlying solutions and programs that are correct solutions to different programming problems.

Task variation is not easily accomplished in the generation strategy because the design and coding of complete programs is heavily emphasized at the cost of other activities; besides, many programming problems are presented but only a few programs actually demonstrate correct problem solutions. On the contrary, task variation is easily accomplished in the completion strategy because all kinds of tasks naturally appear in the course. In addition, almost all problems are presented in combination with a (partial) solution, so that students are confronted with a wide variety of problems as well as solutions to those problems that have the form of well-designed, working programs.

A second advantage of the completion strategy is related to its natural use of examples. This advantage is considered to be particularly important because it directly concerns the transition from passive, declarative knowledge to desired programming behavior. For example, Anderson, Farrell, and Sauers reported that students who are learning to program make a highly selective use of instructional materials: They use worked-out examples of problem solutions that are related to the problem at hand and which have the form of concrete computer programs [19]. Lieberman observed that such "learning by example" has not been taken seriously enough in designing programming instruction and proposes that examples of solutions should be the kernel of well-designed instruction. Students can use such examples as blue-prints to map their new solutions and they may generalize from it to learn new programming principles and techniques.

More specifically, worked-out examples are expected to provide the students with stereotypic sequences of computer instructions or programming language *templates*. Those templates can be related to single program lines (e.g., PRINT "*text*"; *variable*), several program lines (e.g., a looping structure with proper initializations above the loop and counters or running totals placed correctly within the loop), or complete programs (e.g., a general input-process-output template). Thus, whereas low-level templates provide statements to use in one particular situation, higher level templates are applicable in increasingly wider varieties of situations.

Ehrlich and Soloway and Soloway referred to learned templates as "programming plans" and stressed the importance of teaching templates in elementary programming [21, 22]. Once learned, the templates facilitate both the design and coding of programs. In program design, high level templates may be used to separate input, process, and output; then, the problem can be further decomposed into parts that can be performed with other known templates. In the coding of program lines, low level templates may be used to combine elementary commands with their arguments using the correct syntax. Consequently, the availability of correct templates is expected to increase the probability that a semantically correct solution is reached as well as the capability to correctly code this solution.

Students learn templates from worked-out examples that are presented in lectures, textbooks, or practice. In fact, examples appear more and more in textbooks that teach elementary programming (e.g., [23, 24]). In the generation strategy, examples can obviously be presented in class or textbooks; however, they are usually presented in isolation from the tasks assigned as practice, that is, there is never a direct bond between the examples and the program generation tasks. Consequently, students will often skip over the examples, not study them at all, or only start searching for examples that fit in with their solution when they experience serious difficulties in solving a programming problem. In contrast, presenting worked-out examples automatically takes place in the completion strategy. Students are required to study the examples carefully because there is a direct, natural bond between examples and practice: They cannot correctly finish an assignment without studying the program that has to be completed.

In this study, I compared learning outcomes for two groups of high school students that attended an introductory programming course which was designed according to either the completion strategy or the generation strategy. As the main hypothesis, it is stated that the students in the completion group, who perform assignments that offer higher task variation and a direct bond between examples and practice, will be superior in measures concerning program construction because they are more familiar with programming language templates.

In accordance with Mayer and Webb, two more categories of learning outcomes were distinguished: 1) factual, passive knowledge of basic commands and syntax, and 2) ability in interpreting programs [8, 25]. Although these categories are not the ultimate goals of programming instruction, they may be seen as vehicles to reach proficiency in program construction. With regard to knowledge of basic commands and syntax, no difference between the groups is predicted because such elementary knowledge can be learned without having knowledge of templates. However, the completion group is expected to be superior to the generation group in interpreting programs, because they should have greater availability of programming templates which should be useful in program comprehension tasks. The following study was designed to test these hypotheses.

METHOD

Subjects

At the beginning of the scholastic year, students fifteen to seventeen years of age attending a Dutch high school received an entry form to apply for participation in a ten-lesson introductory programming course, which was not part of the students' regular time-table. Thirty-three males and twenty-four females volunteered for participation.

The self-selected sample was divided in two experimental groups by a combined matching procedure [26]. Four variables were used for classification: Sex, year of study, variant of study (exact vs. language and humanities), and prior experience with computers. Two levels were distinguished with regard to prior experience: 1) no experience or only some experience with games and/or applications software, and 2) some experience with both games and/or applications software and programming in a high-level language (usually BASIC). Exactly matched pairs were formed and randomly assigned over the experimental groups; the allotment of remaining subjects was balanced so that the means of the groups were equal for all classification variables. This procedure resulted in one group of twenty-eight subjects and one group of twenty-nine subjects, of which twenty-one pairs were matched exactly.

Materials

Questionnaire — The participating students filled in a short questionnaire two weeks before the start of the course. It contained questions about their age, sex, year of study, kind of attended courses, prior experience with computers, motivation for participation, and so forth.

Classroom setting and programming language — All lessons took place in a computer class equipped with a network of fifteen NewBrain microcomputers. The goal of the course was the acquisition of some elementary programming skills using a small subset of the computer language COMAL-80 [27]. This particular language was chosen for several reasons:

1. Structured programming is supported in a user-friendly environment,
2. The comprehension and evaluation of programs is simplified by allowing only one statement per line,
3. Graphical, Logo-like facilities are provided that students find highly motivating, and
4. Students were not likely to have had experience with the language because it is relatively new and, in The Netherlands, not yet well-known.

Instructional materials — All instruction was based on workbooks that contained five chapters on different subjects [28]. Each chapter included enough information and training materials to provide for two periods of forty-five minutes. A short summary completed each chapter and a comprehensive summary concluded the whole workbook. The main topics in the course were: 1) variables, input, output, and arithmetic operations, 2) loop-exit-endloop structures for iteration, 3) counter variables, running totals, and initializations, 4) simple graphics commands, and 5) integer division and its remainder. The workbooks were available in two versions that profoundly differed in their implemented instructional strategy; one version emphasized the generation of new programs, the other version emphasized the completion of existing programs.

For both versions, the structure of each chapter is displayed in Figure 1. The chapters for both strategies began with a presentation of *factual information* about new commands and syntactical details. The study of this part did not require the use of the computer and lasted approximately thirty minutes. For example, the following issues were dealt with in the first chapter: What is a variable? What is the distinction between the name and the value of a variable? How do you assign a value to a variable? How do you perform arithmetic operations on variables? How do you print the value of a variable or expression on the screen?

For the generation strategy, each chapter consisted of two additional parts. In the first part, a *model solution* was presented that demonstrated the generation of a program which used the newly learned commands and syntax. The study of this part lasted approximately fifteen minutes. Although it was permitted to type in the program to experiment with it, use of the computer was not necessary. For example, the model solution in chapter three dealt with counters and running totals and started from the problem of how to calculate and output the mean of an array of

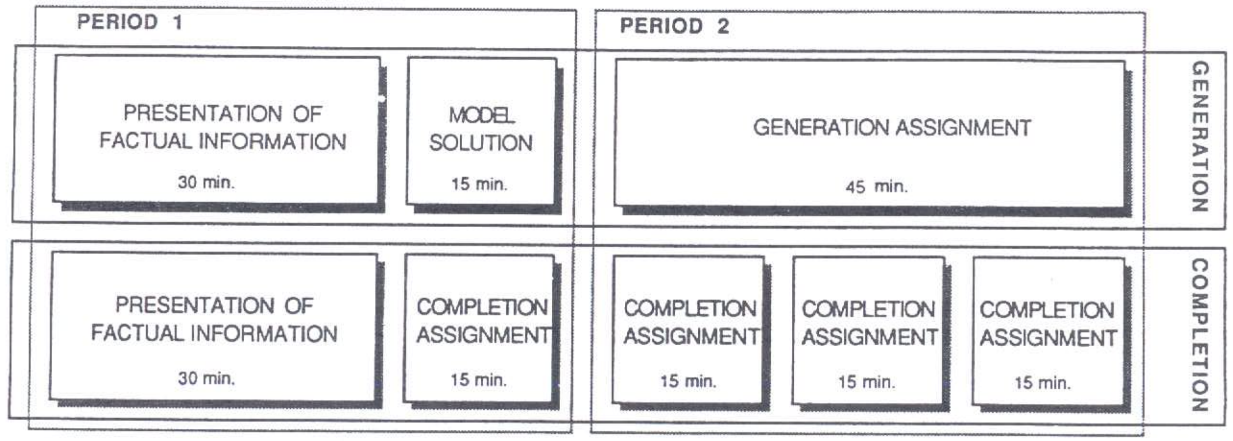

Figure 1. The structure of each of the five chapters for the generation strategy and the completion strategy.

input values. A step-by-step description of the development of a correct program was provided.

In the second part, a *generation assignment* was presented that required the design, coding, and debugging of a complete program. Each assignment offered a problem specification and some guidelines for designing and testing the program. The students were given the entire second period (45 minutes) to complete the assignment. For example, the assignment for chapter four concerned the generation of a program to draw stars with *n* beams. In addition to the number of beams, the length of their sides and their interior angle had to be treated as input variables. The program had to be tested and debugged until it worked correctly.

For the completion strategy, each chapter consisted of only one additional section that contained four *completion assignments*. Each assignment had three components: 1) a problem specification, 2) a complete or, more usually, partial solution in the form of a well-structured program that used the newly learned commands and syntax, and 3) some questions concerning the structure and the working of this program as well as instructions to modify or complete it. The available time to finish each assignment was approximately fifteen minutes so that one problem could be worked on in the first period and the remaining three in the second period. Because it took students considerably less time to finish a completion assignment than a generation assignment it was not possible to present the same amount of problems across conditions. However, the problem specification of one randomly chosen completion assignment was for each chapter identical to the generation assignment in the other version of the workbook.

All (partial) programs presented in the completion assignments were available on disk and could be downloaded from the network. The students answered questions concerning the program's actions in their workbooks; all other tasks required the use of the computer. For example, one of the assignments of chapter four concerned a problem specification and a program to draw polygons. The students had to run this program with several input values to observe and study its workings, they had to modify it in order to draw spirals instead of polygons, and finally they had to complete it by adding extra input variables.

Evaluation forms — The students had to fill in a course evaluation form after the *fifth* and the *last* lesson. The form could be used to comment on the design and content of the course. In addition, it included six closed questions that the students scored on a five-point scale. The questions were related to:

1. The attractiveness of the factual information,
2. The degree of difficulty of the factual information,
3. The attractiveness of the assignments,
4. The degree of difficulty of the assignments,
5. The quantity of information and assignments that had to be studied and performed during the lessons, and
6. The amount of time that was spent studying course-topics at home.

Tests — Three tests were administered to assess learning outcomes: 1) a program construction test to measure skill in designing and coding complete, new programs, 2) a factual knowledge test to measure passive knowledge of commands and syntax, and 3) a program comprehension test to measure proficiency in interpreting programs.

The program construction test consisted of two programming problems for which students had to generate programs on paper; the available time to finish this test was sixty minutes. *Both* programs had to satisfy the following restriction:

1. Only existing commands are used.

Furthermore, both programs required, in addition to some specific characteristics, the use of seven essential COMAL-80 features or programming concepts. These common, necessary features are:

2. Variables, written as single names and a unique variable name for each variable;
3. The input-command, as in the statements <INPUT *variable*> or <INPUT "text": variable>;
4. The print-command, as in the statements <PRINT variable>, <PRINT expression>, <PRINT "text">, <PRINT "text"; *variable*>, or <PRINT "text"; expression>;
5. Conditionals, in the term <*a* relation *b*>, where a and b stand for *variable* or *expression*, and *relation* stands for =, <, <=, >, or >=;
6. Iterations, in the term <LOOP / EXIT WHEN *conditional* / ENDLOOP>, where each of the three elements is coded on a new line;
7. Counter variables or running totals, in a looping structure as <*counter*:= *counter operator constant*>, <*running*:= *running operator variable*>, or <running: = running operator expression>, where operator stands for + or –;
8. Counter variables or running totals initialized above a looping structure, as <*counter*:= *constant* > or <*running*:= *constant*>.

In addition, the readability and usability of both programs could be enhanced by the use of four other characteristics. These optional features are:

9. Print- and input commands combined, as in the statement <INPUT "*text*": *variable*>;
10. Double print commands combined, as in the statement <PRINT "*text*"; *variable*> or <PRINT "text"; expression>;
11. Variable names that reflect their functions;
12. Comments, as <// *comment*> to indicate both the scope of the program and the functions of various parts.

Both the factual knowledge test and the program comprehension test included eighteen four-choice questions; the available time to finish each test was thirty minutes. All items were selected from a small existing item bank and had expected *p*-values of approximately .60. The items of the factual knowledge test evaluated knowledge of single commands and syntactical details of COMAL-80; the items of the program comprehension test referred to the workings and actions of two complete programs that were presented on paper.

Procedure

Formation of groups — The instructional materials were randomly assigned to the experimental groups: One group (n = 28) received the generation workbook and one group (n = 29) received the completion workbook. Subgroups of two or, if necessary, three students were formed that worked together on one computer for all lessons. In case of non-attendance or mortality, the remaining students were assigned to new subgroups so that they always worked in groups of two or three students.

Activities during the course — The instruction was given for one period a week for ten weeks. The two groups attended the course at the same time of day but on different days of the week. All lessons occurred in subsequent weeks, except for a break between lessons four and five because of school vacations. All procedures were identical for both groups.

The first lesson started with a short verbal introduction about the structure and the goals of the course. Then all students received either a generation or a completion workbook. Whereas the workbooks did not contain actual group tasks, the students were encouraged to work together on all assignments. More specifically, they were instructed:

1. To work strictly according to the workbook by answering questions and performing tasks in the indicated sequence,
2. To discuss all difficulties regarding the text or tasks within their subgroup and to consult the teacher only in case of insuperable difficulties, and
3. To work at their own tempo but to follow the time schedule presented in the workbook as closely as possible.

The students in each subgroup regularly switched places to give everyone equal opportunity to enter information on the computer; they answered all questions in their workbooks and performed further tasks by using the computer.

Two experienced teachers conducted the course; each of them led an equal number of lessons for both groups. The teachers' contribution to the course was limited because the subgroups worked independently on their workbooks. The teachers were instructed: 1) to help subgroups in case of serious difficulties that they could not resolve for themselves, and 2) to encourage subgroups to let all students spend an equal amount of time entering information on the computer.

Administration and scoring of tests — The factual knowledge test and the program comprehension test were administered in the week after the last lesson; the program construction test was administered in the subsequent week. Students worked individually on all tests, without assistance of other students or the teacher. The students received a certificate of participation after they finished the program construction test.

The scores on the factual knowledge test and the program comprehension test were determined by the number of correctly answered items. Scoring of the program construction test was conducted in three steps. First, the number of correctly and incorrectly coded lines were counted; incorrectly coded lines were simply defined as lines that would not have been accepted by the COMAL interpreter because of syntactic errors.

Second, two observers scored the correct use of each of the seven COMAL-features or programming concepts that had to be present in both programs, and the restriction that only existing commands were used, as *true* or *false*; in addition, they scored the correct use of each of the four optional features as *true* or *false*. Then, the overall quality of each program was computed as the number of true categories, so that each student's maximum overall score for the correct use of features was twenty-four.

Finally, the two observers scored the semantic correctness of each program on the following five-point scale:

1. The program is hardly recognizable as a "real" program (e.g., no general input-process-output plan is used),
2. The program can be recognized as a "real" program but obviously does not try to reach its goal because the functional units do not perform the required task,
3. The program clearly tries to reach its goal but includes both semantical and syntactical errors,
4. The program is semantically correct but includes syntactical errors, and
5. The program is semantically as well as syntactically correct.

Consequently, each student's maximum score for the semantic correctness of constructed programs was ten.

RESULTS

Mortality and Non-Attendance

In the completion group, twenty-six of the twenty-nine subjects completed the course; in the generation group, only twenty-one of the twenty-eight subjects completed the course. The percentages of mortality in subgroups, classified by sex and prior knowledge, are displayed in Table 1.

Table 1. Percentages of Mortality in Subgroups Classified by Sex and Prior Experience

	Group			
Subgroup	*n*	*Completion*	*n*	*Generation*
Males				
Medium	8	12.5	8	12.5
Low	11	18.2	6	16.7
Females				
Medium	1	—	1	—
Low	9	—	13	38.5*

*p < .05

As may be seen from this table, the difference in mortality is completely accounted for by the subgroup of female subjects with low prior knowledge. In the completion group, all females with low prior knowledge completed the course; in the generation group, only eight of the thirteen female subjects with low prior knowledge completed the course, $\chi^2(1, N = 22) = 4.48$, $p < .05$. No significant differences in non-attendance were observed for subjects that completed the course. The non-attendance rates were 8.5 percent in the completion group and 6.7 percent in the generation group.

After the course, the composition of the experimental groups was no longer equivalent for each classification variable as a result of the differential mortality. Therefore, subsequent analyses on test scores are carried out on matched pairs only; fifteen of the original twenty-one pairs were available after the course was completed.

Tests

The median number of program lines coded for both programs in the program construction test was twenty-four for both groups; the corresponding means were 23.5 for the completion group and 29.5 for the generation group. With respect to the validity of those numbers, it should be recalled that only one statement per line could be coded. The median percentage of incorrectly coded lines was 13.6 percent for the completion group and 24.1 percent for the generation group; the means were, in order, 18.7 percent and 31.8 percent. According to a Wilcoxon signed rank test, this difference is significant, $W(15) = 26$, $p < .05$; the Hodges-Lehmann estimator for the effect size is thirteen.

Table 2 displays the proportions of programs in which the essential and optional features of the programming language COMAL-80 were used correctly. The completion group was superior to the generation group for all features (significantly on the 10% level for five features), with exception of the use of conditionals (e.g., in EXIT WHEN conditional). The coefficient alpha for the internal consistency of the measure = .83; the interobserver reliabilities ranged from .69 to 1.00. A signed

Table 2. Proportions of Programs in which Necessary and Optional Features were Used Correctly

	Group ($N = 15$)			
Features	*Completion*	*Generation*	r_{obs}	*W*
NECESSARY				
1. Commands	.83	.60	.77	[a]28.5*
2. Variables	.87	.67	.71	39.0
3. Input	.70	.47	1.00	33.0*
4. Print	.73	.63	.98	46.5
5. Conditionals	.47	.60	.91	87.0
6. Looping	.53	.47	.90	53.0
7. Counters	.53	.33	.96	29.5**
8. Initialization	.43	.27	.92	32.5*
OPTIONAL				
9. Print /Input	.20	.13	1.00	53.0
10. Print /Print	.37	.20	1.00	39.5
11. Names	.93	.73	.69	39.0
12. Comments	.53	.17	.89	23.0***

[a]*N* is fourteen because an odd number of zero differences occurred.
*$p < .10$.
**$p < .05$.
***$p < .025$.

rank test was conducted to test the significance of the differences. The most conspicuous difference occurred in the use of comments. In the completion group, 53 percent of the constructed programs contained meaningful comments; in the generation group, only 17 percent of the programs contained such comments.

With regard to the overall scores for the correct use of features, a maximum score of twenty-four points could be reached if a subject correctly used all twelve features in both programs. The interobserver reliability for those scores = .98. The medians were fifteen for the completion group and ten for the generation group; the corresponding means were 14.3 and 10.5. A signed rank test yields a significant difference, $W(14) = 13, p < .01$; the Hodges-Lehmann estimator for the effect size is three.

Concerning the semantic correctness of the programs, the subjects could reach a maximum score of ten points if both constructed programs were semantically as well as syntactically correct. The interobserver reliability for these scores = .86. The completion group was superior to the generation group: The medians were, in order, six and five; the corresponding means were 5.6 and 4.8. This difference is significant, $W(15) = 26$, $p < .05$; the Hodges-Lehmann estimator for the effect size is one.

With regard to the factual knowledge test and the program comprehension test, no significant differences were observed between the groups. The medians for the numbers of correctly answered items in the factual knowledge test (KR20 = .75) were seven in the completion group and ten in the generation group; the means were, in order, 8.4 and 10.0. This difference is nonsignificant, $W(14) = 31.5$. The medians for the numbers of correctly answered items in the program comprehension test (KR20 = .54) were eight for both groups; the corresponding means were 8.5 for both groups.

Evaluation

Evaluation data were analyzed for all subjects that completed the course by one-way analyses of variance with repeated measures. The mean scores over the measurements after the fifth and the last lesson are displayed in Table 3.

No significant effects were observed on judgment scores for attractiveness and difficulty of the—for both groups identical—presented factual information or the—completion vs. generation—assignments. However, the (Group X Measurement) interaction for the judged difficulty of assignments nearly yielded a significant effect. The mean scores for the completion assignments were 2.77 on the first measure and 2.85 on the second measure; the mean scores for the generation assignments were, in order, 2.48 and 2.95, $F(1,45) = 3.61, p < .06$.

For the drop outs in the generation group who did complete the first five lessons ($n = 4$), the judgments of difficulty of the generation assignments significantly differed from the rest of the group on the first measure: The means were 3.75 for the drop outs and 2.48 for the rest of the group, $T(23) = 2.20, p < .05$. The judgments of difficulty of assignments in the completion group did not significantly differ between drop outs ($n = 3$) and the rest of the group: The means were, in order, 3.00

Table 3. Mean Judgment Scores of Both Groups for Attractiveness, Difficulty, and Quantity

	Group	
Judgment	*Completion (N=26)*	*Generation (N=21)*
Attractiveness [a]		
Information	2.73	2.74
Assignments	2.29	2.36
Difficulty [b]		
Information	2.56	2.67
Assignments	2.81	2.72
Quantity		
Lessons [c]	3.60	3.29*
Homework [d]	1.75	1.84

[a] 2 = attractive, 3 = neutral.
[b] 2 = easy, 3 = neutral.
[c] 3 = neutral, 4 = quite a lot.
[d] 1 = 0–1/2 hour, 2 = 1/2–1 hour.
*$p < .10$.

and 2.77. With respect to further evaluation data, the drop outs did not show differences to the rest of the groups.

Finally, both groups voluntarily spent the same amount of time on homework. However, the groups slightly differed in their judgment of the amount of work that had to be done during the lessons: Mean scores for the completion group and the generation group were, in order, 3.60 and 3.29, $F(1,45) = 3.09, p < .08$. After the last lesson, all subjects were asked to estimate the percentage of information and assignments that could be thoroughly studied and exercised during the course. These estimations yielded mean scores of 60 percent for the completion group and 84 percent for the generation group, $T(45) = 4.18, p < .005$.

DISCUSSION

The main hypothesis of this study, which stated that the completion strategy is superior to the generation strategy for learning outcomes regarding a program construction task, is clearly supported by the results. First, while the length of constructed programs was statistically equal for both groups, the percentage of correctly coded program lines was higher for the completion group. Second, all but one of the programming language features that had to be present in the constructed programs were more often correctly used in the completion group than in the generation group; furthermore, the overall quality of programs, as measured by the numbers of correctly used features, was higher in the completion group. Finally, the semantic correctness of the constructed programs was superior in the completion group.

The students judged both strategies as equally difficult and equally attractive. In addition, the rate of non-attendance was comparable and the students spent, besides the lessons, the same amount of time to working at home. Consequently, the differential effects of instructional strategies on learning outcomes in program construction cannot be easily accounted for by factors related to affect towards the course or time-on-task.

The superiority of the completion strategy was hypothesized because the higher task variation and, in particular, the direct bond between practice and the presentation of worked-out examples was expected to facilitate the development of templates. The results support this view. As predicted, no difference between the groups could be observed in knowledge of single commands and syntax; in fact, the generation group was slightly superior to the completion group on the factual knowledge test. Nevertheless, the percentage of correctly coded lines was higher for the completion group. This may be explained by the availability of templates that offer structures to code either statements or single lines, that is, to combine basic commands with their arguments using the correct syntax. Extra support for this explanation is given by the fact that non-existing commands and incorrect input-statements appear significantly less frequently in the completion group.

The results also indicate that students in the completion group had better templates available to code several related program lines. For instance, both groups correctly used structures for iteration with the same frequency. However, in the completion group those looping structures were more often combined with proper initializations above the loop as well as a correct use of counter variables and/or running totals within the loop. Obviously, the completion strategy yields better developed templates to combine iterations with related concepts.

Finally, the completion strategy led to better developed general templates of what a well-coded program should look like. For instance, comments to indicate the scope of the program and the goals of certain parts were more common in the completion group. Other high level templates may be used to separate input, process, and output; subsequently, the problem can be decomposed in parts that can be performed with lower-level templates. This process should increase the probability that a semantically correct solution is found, thus, it may explain the finding that better solutions were reached in the completion group.

Apart from being useful in the construction of programs, templates are expected to be helpful in interpreting programs. However, contrary to the prediction, no difference in test scores were observed on the program comprehension test. One possible explanation is that students learn templates as *units* that serve as both partial solutions to decomposed problems and building blocks to construct programs. However, they do not really seem to understand the working of those units, that is, the flow of control within them.

As a speculation, it may be that correctly using templates comes prior to understanding their internal workings. In this case, the short duration of the course may account for the absence of a difference in program comprehension scores. This effect may be further strengthened by the fact that, according to the students' estimations, a smaller amount of the presented information and assignments could be thoroughly studied in the completion strategy. A better balanced volume of instructional materials would probably have enlarged differences in learning outcomes.

In future experiments concerning the completion strategy, measures will be taken to secure a more thorough learning of the flow of control within templates. As Anderson, Boyle, Corbett, and Lewis, as well as Lieberman pointed out, examples should be *annotated* with information about what they are supposed to illustrate [20, 29]. The programs presented in the completion assignments can easily be amplified with such remarks. First, it may be desirable to further annotate the examples by explicitly referring to the templates they use. Second, both the critical features of the templates and the flow of control within them should be emphasized to stimulate the gaining of understanding about how they work.

Whereas the results on learning outcomes may be adequately explained by the higher task variation and the direct bond between practice and worked-out examples in the completion assignments, which in turn led to a greater availability of templates in the completion group, it should be noted that other factors may be of importance. In general, instructional strategies such as the generation and the completion strategy can be thought of as *sets* of instructional methods, or tactics, that pertain to the design of practice (e.g., kind and duration of assignments and the quality of feedback) and instructional materials (e.g., presentation of factual information and model solutions) [17]. Although the generation group and the completion group most profoundly differed in the kind of assignments they performed (i.e., generation assignments that offer little task variation and no worked-out examples vs. completion assignments that offer more task variation and a natural bond between examples and practice), they also differed in other respects.

First, the groups differed in time spent doing assignments. The completion group spent sixty minutes per lesson working on completion assignments and the generation group spent forty-five minutes per lesson working on generation assignments. Whereas it can be argued that both groups spent approximately the same amount of time to actual programming tasks because the completion assignments involved a substantial amount of program reading, which is the same activity as the generation group was involved in while studying the model solutions, the possibility that the different amounts of practice influenced learning outcomes cannot with certainty be precluded.

Second, no quantitative data are available on the feedback that students received. Apart from the successful execution of the programs, both groups received feedback from the teacher in case of difficulties they could not resolve for themselves. In making use of the feedback of the COMAL interpreter, the generation group may have been at a disadvantage because they would have more self-generated code and thus it could be more difficult for them to identify the code responsible for the "buggy" output. But as a consequence, more help from the teacher would be expected in the generation group and actually, both teachers confirmed this expectation. However, the contribution of eventual differences in the amount and quality of feedback to learning outcomes cannot be easily determined.

Third, the strategies differed in their presentation of instructional materials because the generation group received model solutions and the completion group did not. Although the completion group had (partial) programs available on disk and were required to observe the execution of programs while working on the completion assignments, the generation group were only given a printout of the model solutions in their workbooks. It is possible that observing the dynamic execution of a program facilitates understanding of programming concepts to a greater degree than reading a program. In fact, it is possible to study learning outcomes for more similar strategies by presenting the model solutions on line for the generation strategy.

Generally, in future research on instructional strategies it will be necessary to make further careful comparisons of learning outcomes for strategies that apply different sets of tactic for the design of practice and the presentation of instructional materials. This research should not only focus on tracking down global guidelines for instructional design, but also on an assignment of weights to instructional tactics and an assessment of possible interactions between tactics. Then, it will become more clear which tactics must be included in instructional strategies that contribute to higher learning outcomes.

Besides effects on learning outcomes, a conspicuous and unexpected difference between the completion and the generation strategy occurred with regard to mortality in the groups. In the completion group, mortality was—for a self-imposed, supplementary course—relatively low and equally divided over subgroups; in the generation group, mortality was higher and primarily occurred for female subjects with low prior knowledge. Furthermore, the drop outs in the generation group judged the assignments as difficult compared to the rest of the group, whereas the drop outs in the completion group did not differ in their judgments from the rest of the group. Possibly, female subjects with low prior knowledge experience the generation of complete programs as a difficult and menacing task: The completion strategy largely obviates this difficulty.

A second result concerning the difficulty of assignments indicated that in the generation strategy the judged difficulty of assignments increased during the course, whereas in the completion strategy assignments were judged equally difficult throughout the course. These judgments of difficulty can be related to required processing load [17]. In the generation strategy, task-required processing load rises during the course as programs have to be generated for increasingly difficult programming problems; processing overload is hard to prevent because the design and coding of new, complete programs is the primary student activity. On the contrary, task-required processing load is more easily controlled in the completion strategy because this strategy varies the complexity of student activities, such as reading, tracing, modification, and completion: The stable judgment of the difficulty of completion assignments supports this view.

Summarizing, the completion strategy seems to be an excellent alternative to more traditional strategies that mainly emphasize the generation of programs for three reasons:

1. It results in better learning outcomes for program construction,
2. It is characterized by a lower mortality rate, in particular for female students with low prior knowledge about computers, and
3. There is reason to believe that it shows superior control over students' task-required processing load in working on programming assignments during the course.

With regard to the results, it finally should be explained why the reported study was primarily interested in performance measures and not in transfer of learned skills. Although one of the most often cited rationales for programming courses is that learning computer programming will improve students' higher level, domain independent cognitive skills, there is conflicting research evidence regarding the relationship between computer programming and such skills. Low learning outcomes in most programming courses may account for the conflicting results because the cognitive skills that are expected to develop out of programming no doubt depend upon attaining a certain proficiency in programming by the students. As Kurland et al. pointed out, we need to more closely study instructional strategies that yield higher learning outcomes before we prematurely go looking for far transfer effects from programming; the present study was designed to provide for this need [2].

In future research concerning the completion strategy, more attention will be paid to measures that secure a more thorough understanding of the flow of control within templates. Then, two paths will be taken. First, in addition to learning outcomes, the problem solving processes and problem approaches of students who are working according to the completion or the generation strategy will be observed. For this purpose, parts of both strategies will be implemented in the form of computer-assisted instruction. Close observations of individual students working according to one of the strategies will yield further, more precise information about the appropriateness of the completion strategy.

Second, the completion strategy will be implemented in a regular high school programming course so that learning outcomes can be studied for larger groups that compulsory participate in a course of relatively long duration. Unfortunately, much effort will have to be invested in designing such a course as yet few instructional materials are available that could be classified under the completion strategy. However, this investment is believed to be worth while as the results of the present study consolidated the view that a change in instructional strategies is needed to reach more satisfactory learning outcomes, and could thus lead to eventual far transfer effects of learned cognitive skills.

ACKNOWLEDGMENTS

I thank Sanne Dijkstra, Otto Jelsma, and Hein Krammer for their helpful comments on a draft of this article.

REFERENCES

1. J. Dalbey and M. C. Linn, The Demands and Requirements of Computer Programming: A Literature Review, *Journal of Educational Computing Research, 1*:3, pp. 253–274, 1985.
2. D. M. Kurland, R. D. Pea, C. Clement, and R. Mawby, A Study of the Development of Programming Ability and Thinking Skills in High School Students, *Journal of Educational Computing Research,* 2:4, pp. 429–485, 1986.

3. M. C. Linn, The Cognitive Consequences of Programming Instruction in Classrooms, *Educational Researcher, 14*:5, pp. 14–29, 1985.
4. M. C. Linn, K. D. Sloane, and M. J. Clancy, Ideal and Actual Outcomes from Precollege Pascal Instruction, *Journal of Research in Science Teaching, 24*:5, pp. 467–490, 1987.
5. R. D. Pea and D. M. Kurland, On the Cognitive Effects of Learning Computer Programming, *New Ideas in Psychology, 2*:2, pp. 137–168, 1984.
6. D. N. Perkins, C. Hancock, R. Hobbs, F. Martin, and R. Simmons, Conditions of Learning in Novice Programmers, *Journal of Educational Computing Research, 2*:1, pp. 37–56, 1986.
7. P. R. Pintrich, C. F. Berger, and P. M. Stemmer, Students' Programming Behavior in a Pascal Course, *Journal of Research in Science Teaching, 24*:5, pp. 451–466, 1987.
8. R. E. Mayer, Different Problem-Solving Competencies Established in Learning Computer Programming with and without Meaningful Models, *Journal of Educational Psychology, 67*:6, pp. 725–734, 1975.
9. R. E. Mayer, The Psychology of How Novices Learn Computer Programming, *Computing Surveys, 13*:1, pp. 121–141, 1981.
10. R. E. Mayer, Contributions of Cognitive Science and Related Research in Learning to the Design of Computer Literacy Curricula, in *Computer Literacy*, R. Seidel, R. Anderson, and B. Hunter (eds.), Academic Press, New York, pp. 129–159, 1982.
11. R. E. Mayer and B. Bromage, Different Recall Protocols for Technical Text Due to Advance Organizers, *Journal of Educational Psychology, 72*:2, pp. 209–225, 1980.
12. J. M. Hoc, Planning and Direction of Problem Solving in Structured Programming: An Empirical Comparison between Two Methods, *International Journal of Man-Machine Studies, 15*:4, pp. 363–383, 1981.
13. S. McCoy Carver and D. Klahr, Assessing Children's Logo Debugging Skills with a Formal Model, *Journal of Educational Computing Research, 2*:4, pp. 487–525, 1986.
14. L. E. Deimel and D. V. Moffat, A More Analytical Approach to Teaching the Introductory Programming Course, in *Proceedings of the NECC*, J. Smith and M. Schuster (eds.), The University of Missouri, Columbia, pp. 114–118, 1982.
15. J. Dalbey, F. Tourniaire, and M. C. Linn, *Making Programming Instruction Cognitively Demanding: An Intervention Study*, ACCCEL Report, University of California, Berkeley, 1985.
16. R. D. Pea, Language Independent Conceptual "Bugs" in Novice Programming, *Journal of Educational Computing Research, 2*:1, pp. 25–36, 1986.
17. J. J. G. Van Merriënboer and H. P. M. Krammer, Instructional Strategies and Tactics for the Design of Introductory Computer Programming Courses in High School, *Instructional Science, 16*:3, pp. 251–285, 1987.
18. R. Brooks, Towards a Theory of the Cognitive Processes in Computer Programming, *International Journal of Man-Machine Studies, 9*:6, pp. 737–751, 1977.
19. J. R. Anderson, R. Farrell, and R. Sauers, Learning to Program in LISP, *Cognitive Science, 8*:2, pp. 87–129, 1984.
20. H. Lieberman, An Example Based Environment for Beginning Programmers, *Instructional Science, 14*:3, pp. 277–292, 1986.

21. K. Ehrlich and E. Soloway, An Empirical Investigation of The Tacit Plan Knowledge in Programming, in *Human Factors in Computer Systems*, J. Thomas and M. L. Schneider (eds.), Ablex Publishing Corporation, Norwood, New Jersey, pp. 113–133, 1984.
22. E. Soloway, From Problems to Programs via Plans: The Content and Structure of Knowledge for Introductory LISP Programming, *Journal of Educational Computing Research, 1*:2, pp. 157–172, 1985.
23. N. Dale and D. Orschalick, *Introduction to Pascal and Structured Design*, D. C. Heath and Company, Lexington, 1983.
24. D. V. Moffat, *Common Algorithms in Pascal*, Academic Press, New York, 1984.
25. N. M. Webb, Microcomputer Learning in Small Groups: Cognitive Requirements and Group Processes, *Journal of Educational Psychology, 76*:6, pp. 1076–1088, 1984.
26. H. P. M. Krammer, *IND1.PAS en IND2.PAS: Twee programma's voor het matchen als exacte matching onmogelijk is* [IND1.PAS and IND2.PAS: Two programs for matching when exact matching is impossible], Technical Report Number IST-MEMO-86-03, University of Twente, Enschede, The Netherlands, 1986.
27. B. R. Christensen, *Beginning COMAL*, Ellis Horwood, Chichester, 1982.
28. J. J. G. Van Merriënboer, *Leren programmeren in COMAL: Experimentele werkboeken 1 en 2* [Learning to program in COMAL: Experimental workbooks 1 and 2], University of Twente, Enschede, The Netherlands, 1987.
29. J. R. Anderson, C. F. Boyle, A. Corbett, and M. Lewis, *Cognitive Modelling and Intelligent Tutoring*, Technical Report Number ONR-86-1, Carnegie-Mellon University, Pittsburgh, 1986.

Direct reprint requests to:

Dr. Jeroen J. G. van Merriënboer
Department of Instructional Technology
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands