Pedagogical Explanation Methods of Teaching Matrix Programming Operations: Effects on Students' Achievements in IPCs

Assoc. Prof. Dr. Arjan Skuka
Izmir University, Turkey
Department of Software Engineering
arjan.skuka@izmir.edu.tr

Abstract

Despite the fact that introductory programming courses (IPCs) are taught at universities for more than thirty years, students still find computer programming very difficult to learn. Programming pedagogy deals with the methods and principles of teaching and learning computer programming. The programming pedagogical approaches that have been proposed to increase the efficiency of teaching and learning computer programming mostly focus on the tools, paradigms, programming languages and environments used in IPCs. To increase significantly the students' success rates in IPCs, these approaches should be complemented with pedagogical explanation (PE) methods. This research is focused on a PE method of teaching sequential search of a matrix row (SSMR). The research was designed as experimental study with pretest-posttest control group model, involving students of Computer Engineering department Izmir University. While the experimental group was subjected to a pedagogical explanation method, a traditional explanation method was applied in the control group. To collect the research data, an achievement pretest, posttest and a questionnaire were developed and applied. The research findings showed the effectiveness of teaching SSMR by using a PE method. This method positively influenced students' level of topic comprehension, which consequently improved their achievements. In order for students to understand better the other matrix programming operations, similar PE methods should be developed and used in IPCs. On a more general level, the results of this research suggested that PE methods should be developed and used for other topics that students usually find difficult to understand in IPCs. Using these methods can be a very important factor in significantly increasing students' success in IPCs.

Keywords: Programming Pedagogy, Introductory Programming, Pedagogical Explanation Method, Matrix Programming

1. Introduction

Although *introductory programming courses* (IPCs) are taught at universities for more than thirty years, students still find computer programming very difficult to learn. This situation has had negative consequences on Computer Science (CS) education. In many developed countries, even though the demand for CS specialists of different profiles is increasing, the number of students in these study programs is steadily decreasing (Mason, Cooper and de Raad, 2012; Vitkute and Vidžiunas, 2012). Some of the main reasons for this downward trend is curriculum complexity, its insufficient links with practical needs and extremely rapid IT evolution. It has been stated numerous times in various research papers that learning programming is a difficult task to achieve (Brown, 2006; Chetty and Barlow-Jones, 2014; Mason and others 2012; Saeli, Perrenet, Jochems and Zwaneveld, 2011; Vujosevic and Tosic, 2008). Research papers continuously reveal high failure and dropout rates for IPCs (Bennedsen and Caspersen 2007, Chetty and others, 2014; Fares and Fares, 2014; Robins, 2010). This is partially a result of the recent trend of enrollment of non-CS major students in IPCs. The most significant reason for these high failure and dropout rates, yet again is related to the difficulties that students face when exposed to complex programming topics.

Programming pedagogy, as one of the emerging research areas in computer science, deals with the methods and principles of teaching and learning computer programming. A variety of programming pedagogical approaches have been proposed to increase the efficiency of learning computer programming. Mostly, they focus on the tools, paradigms, programming languages and environments used in IPCs (Hadjerrouit, 2008; Meyer, 2003; Porter and Simon, 2013; Stephen, Elizabeth, Ogao, Franklin and Ikoha, 2012; Woszczynski, Haddad and Zgambo, 2005; Zingaro, Bailey Lee, and Porter, 2013). These approaches have had positive effects in IPCs, but unfortunately they did not solve the problem. Introductory programming remained complex and difficult for majority of students to learn.

1.1 Pedagogical Explanation Methods

To increase significantly the students' success rates in IPCs, previously mentioned approaches should be complemented with optimal topic explanation methods. *How* a programming topic is explained is crucial to the level of students' comprehension of the topic. Being an expert in programming doesn't make an instructor a good teacher by default. Unfortunately, a considerable number of instructors explain complex programming topics to students in less understandable ways. Some of the reasons for that, among others, are:

- some instructors do not possess an appropriate programming pedagogy background;
- some instructors are neglecting the fact that majority of students in IPCs have not yet developed the "programming logic" and cannot understand their way of explanation;
- considerable number of textbooks for IPCs used by instructors are not as "good" as they should be − they miss pedagogically based explanations of programming topics;
- some instructors are being focused on research and are not spending enough time on preparing pedagogically based lectures.

Based on our experience in teaching IPCs in C, C++, Java, Scheme and Racket at several universities, explaining programming topics in a step-by-step and understandable way is a very important factor in increasing students' success in IPCs. This way of teaching represents a solid basis for the other student activities in the course: lab exercises, homework, programming projects, solving problems using Online Judges etc. On the other hand, if the topic is explained in less understandable way, it usually frustrates and demotivates students for the required further activities.

A *pedagogical explanation method* is a well planned and structured sequence of steps to present a programming topic in a concise, clear and understandable way. Recursion, matrix programming operations (MPOs) and linked lists are some of the topics that students usually find difficult to understand in IPCs. In this paper we have focused on explanation methods of teaching one fundamental MPO: *sequential search of a matrix row* (SSMR). This operation is easy for experienced programmers, but it is difficult to learn for beginners in programming. The purpose of the research was to investigate the effects of two different explanation methods on students' level of topic comprehension as a result of using a *traditional explanation* (TE) method and a *pedagogical explanation* (PE) method.

2. Method

Research design

In this research, we have used an experimental study with pretest-posttest control group model aimed at measuring the effects of two different explanation methods on students' level of topic comprehension. In accordance with this model, *control group* (CG) and *experimental group* (EG) were created and experimental lectures using two different explanation methods were conducted.

Research sample

Participants in the experiment were 60 students of Computer Engineering department in Izmir University attending the Algorithms and Programming II course, which focuses on procedural programming in C. Both CG and EG were consisted of 30 students. The distribution of students in the groups was done based on their grades in Algorithms and Programming I course[1], insuring that there was no significant difference between the total average grades of students in both groups.

[1] The course focuses on functional programming in Racket. According to us, functional programming followed by procedural programming is not an optimal approach for IPC. It is quite a rare approach in the world, as well as in Turkey. But, since it is the current curriculum policy in our faculty we have to follow it, despite our believes.

Research instrument and procedure

To collect the data of the study, two tests and one questionnaire were developed. In order to determine the students' previous knowledge of SSMR, a pretest was conducted for both groups. After that, we conducted experimental lectures using two different explanation methods in the spring semester of the 2013-14 academic year. SSMR was taught to students in CG by using a traditional explanation method, while our pedagogical explanation method was used in EG. The survey process was finished by conducting a posttest and a questionnaire to students of both groups.

Data Analysis

Students' level of a programming topic comprehension can be measured in terms of *knowledge* level and *application* level (Davies, 1993). The *achievement pretest* contained one programming problem aimed at determining the students' previous knowledge of SSMR. The *achievement posttest* contained two programming problems aimed at determining students' knowledge and application levels of SSMR, after the experimental lectures were conducted. In addition to the two tests, we prepared an *attitude questionnaire* which contained two multiple-choice questions aimed at collecting students' opinions towards both explanation methods. Students' answers were measured using a five-point Likert scale.

The quantitative data from the achievement pretest and posttest was analyzed using the programming language R (R development core team, 2012). we wrote an R script to read students' scores in R data frames and analyze them. For analysis purposes we used independent variables *t-test* and other tools of the language (built-in functions *qt*, *sd*, *barplot* etc.). The level of statistical significance was taken as 0.05.

3. Explanation Methods of Teaching SSMR

In this section, a traditional and a pedagogical explanation method of teaching SSMR are described. SSMR problem statement: Write a function to find a given number in a specified unsorted row of a matrix of integers. If the number is found in the row, then the function should return the position of its first appearance (col. index). Otherwise, it should return -1.

3.1 Traditional explanation method

TE method of teaching SSMR consists of two steps: writing a function for SSMR and using it in a program.

Step1. Writing a function for SSMR.

The function has four parameters: *matrix* − reference to a matrix, *ncols* − number of columns, *number* – searched number, *row* − index of the row to be searched. It should return the index of the first appearance of the number in the row with index *row*, if it is found. Otherwise, it should return -1. Searching is done by comparing each element of the specified row against the searched number until a match is found or all elements are unsuccessfully compared. If the number is found, the first *return* statement will return the index of the found element and it will exit the function. If the number is not found, then the program will exit the *for* loop. In that case, we need a second *return* statement to return -1 and exit the function.

```
int find_in_row(int matrix[][M], int ncols, int number, int row) {
  int j;
  for(j = 0;  j < ncols;  j++)
    if(matrix[row][j] == number) return j;
  return -1;
}
```

Step 2. Using the function in a program.

```
int n = 4, A = {{3,1,7,5}, {8,4,6,2}, {6,9,0,3}};
int num, row, index;
num = 6, row = 1;     index = find_in_row(A, n, num, row);
num = 0;              index = find_in_row(A, n, num, row);
```

First, the value of *index* is 2 because 6 is found in column 2 of row 1. After the second function call, it will change its value to -1 because 0 is not found in row 1.

3.2 Pedagogical explanation method

In order for students to understand better the programming solution to SSMR, our pedagogical explanation method is based on two important pedagogical techniques:

- Visual tracing of the matrix operation by using concrete input values and specifying matrix cell references for all possible scenarios.
- Inductive derivation of the matrix function from the corresponding array function.

A row of a matrix is an array. The algorithm to search a matrix row is based on the algorithm to search an array. Consequently, our PE method of teaching SSMR is comprised of the following steps:

1. Writing a function to search an array.
2. Visually tracing the SSMR operation.
3. Deriving the function for SSMR from the function to search an array.
4. Using the function in a program (identical to step 2 of TE method).

Step 1. Writing a function to search an array.

The function has three parameters: *array* – reference to an array, *N* – number of array elements, *number* – searched number. It returns the index of the first appearance of the number in the array, if it is found. Otherwise, it returns -1.

```
int find_in_array(int array[], int N, int number) {
  int i;
  for(i = 0;  i < N;  i++)
    if(array[i] == number) return i;
  return -1; }
```

Step 2. Visually tracing the SSMR operation.

In order for students to understand better the SSMR operation, it should be visually presented and traced by using concrete input values and specifying matrix cell references. There are two possible scenarios: finding and not finding the number.

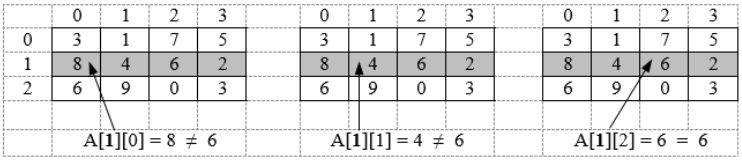

Figure 1. Successful search of the number 6 in row with index 1

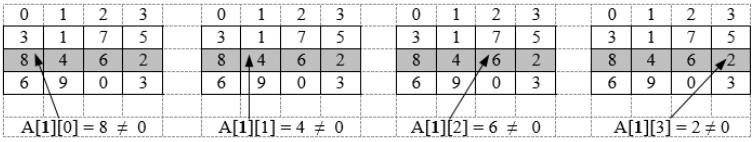

Figure 2. Unsuccessful search of the number 0 in row with index 1

Step 3. Deriving the function to compute SSMR from the function to search an array.

We modify the function *find_array* considering that we have to search a given matrix row. So, we have to select a given row of a matrix and apply the algorithm to search an array elements. To select the index of the row we will use a variable *row*. The variable *array*[i] is modified to *matrix*[row][i] to access the row with index *row* of the matrix. Selecting the elements of the row is accomplished by using the counter *i*.

Searching an array:

```
for(i = 0;  i < N;  i++)
  if(array[i] == number) return i;
```

Searching a matrix row:

```
for(i = 0;  i < N;  i++)
  if(matrix[row][i] == number) return i;
     fixed ↙          ↘ variable (0, 1, ... , N−1)
```

The reference to an array must be replaced with a reference to a matrix in the header of the function and the parameter *row* must be included in the function parameter list. To make the function more understandable to students, we can rename *find_in_array* to *find_in_row*, *N* to *ncols* and *i* to *j* because usually, but not necessarily, *i* is used to select rows and *j* is used to select columns. The final version of the function is identical to the function we wrote using the TE method.

4. Findings and Results

4.1 Achievement Pretest Results and Analysis

To determine students' previous knowledge level of SSMR, they were given the following problem in the achievement pretest: "Write a function to find a given number in a specified unsorted row of a matrix of integers. If the number is found in the row, then the function should return the position of its first appearance (column index). Otherwise, it should return -1". Students' solutions were weighted with maximum 100 points. The obtained findings are given in Table 1.

Table 1

Achievement Pretest Scores of EG and CG Groups

Group	N	$\overline{x}$	sd	t	p
EG	30	9.67	7.98	0.33	0.741
CG	30	10.33	7.54		(p > 0.05)

Table 1 revealed that there was *no significant difference* between the scores of the students of experimental group and control group in the achievement pretest. The mean scores of the students in EG and CG were 9.67 and 10.33, respectively. The obtained t-value was smaller than the critical value (0.332 < 2.001) and the p-value was larger than 0.05 (0.741 > 0.05).

4.2 Achievement Posttest Results and Analysis

To determine students' knowledge and application levels of SSMR after the experimental lectures were conducted, the achievement posttest contained two programming problems. Solutions to each of the problems were weighted with maximum 50% of the total number of points in the test.

Analysis of Scores of EG and CG in the Achievement Posttest Regarding Students' Knowledge Level of SSMR.
To determine students' knowledge level of SSMR, the following problem was given in the achievement posttest: "Write a function to count the number of matrix rows which contain a specified number". The obtained findings are given in Table 2.

Table 2

Achievement Posttest Scores of EG and CG Regarding Students' Knowledge Level of SSMR

Group	N	$\overline{x}$	sd	t	p
EG	30	74.50	18.54	2.67	0.009
CG	30	61.17	20.07		(p < 0.05)

Table 2 shows a significant difference between the scores of the students of EG and CG in the achievement posttest regarding students' knowledge level of SSMR, in favor of the experimental group. The mean score of the students in EG ($\overline{x}$ = 74.50) was 21.79% higher than the mean score of the students in CG ($\overline{x}$ = 61.17). The obtained t-value (2.67) was larger than the critical value (2.001) and the p-value (0.009) was smaller than 0.05.

Analysis of Scores of EG and CG in the Achievement Posttest Regarding Students' Application Level of SSMR.
To determine students' application level of SSMR, the following problem was given in the achievement posttest: "Write a function to sort the numbers of a specified matrix row". The obtained findings are given in Table 3.

Table 3

Achievement Posttest Scores of EG and CG Regarding Students' Application Level of SSMR

Group	N	$\bar{x}$	sd	t	p
EG	30	71.33	17.90	3.30	0.001
CG	30	55.17	19.98		(p < 0.05)

Table 3. shows a significant difference between the scores of the students of EG and CG in the achievement posttest regarding students' application level of SSMR, in favor of the experimental group. The mean score of the students in EG ($\bar{x}$ = 71.33) was 29.29% higher than the mean score of the students in CG ($\bar{x}$ = 55.17). The obtained t-value (3.30) was larger than the critical value and the p-value (0.001) was smaller than 0.05.

Analysis of the Overall Scores of EG and CG in the Achievement Posttest. To determine the overall success of the students in the achievement posttest, we summed the points gained for the solutions to programming problems for each student and computed the mean scores for both EG and CG. The obtained findings are given in Table 4.

Table 4

Achievement Posttest Overall Scores of EG and CG

Group	N	$\bar{x}$	sd	t	p
EG	30	72.92	18.08	3.01	0.003
CG	30	58.17	19.80		(p < 0.05)

Table 4 shows a significant difference between overall mean scores of the students of EG and CG in the achievement posttest, in favor of the experimental group. The mean overall score of students in EG ($\bar{x}$ = 72.92) was 25.36% higher than the mean overall score of the students in CG ($\bar{x}$ = 58.17). The obtained t-value (3.01) was larger than the critical value and the p-value (0.003) was smaller than 0.05.

4.3 Attitude Questionnaire Results and Analysis

Based on the fact that the students in EG obtained higher scores than the students in CG in the achivement posttest, we wanted to determine opinions of both EG and CG students towards both explanation methods. In order to do that it was necessary for the students in each group to be familiar with the explanation method used in the other group. So, we presented the TE method to students in EG and PE method to students in CG. The following two questions were addressed to students in the atittude questionnaire:

1. "Pedagogical explanation method of teaching SSMR is more efficient compared to traditional explanation method in terms of students' level of topic comprehension".

2. "Pedagogical explanation method of teaching SSMR is more understandable to students compared to traditional explanation method".

Students' answers to both questions were measured using a common five-point Likert scale (Strongly Dissagree, Dissagre, Neutral, Agree, Strongly Agree). The obtained findings are given in Table 5.

Table 5

Questionnaire Results Regarding Students' Attitude Towards Explanation Methods

Strongly disagree		Disagree		Neutral		Agree		Strongly agree	
f	%	f	%	f	%	f	%	f	
Question 1 (EG, CG / N = 60)									
0	0	0	0	11	18.33	29	46.67	20	33.33
Question 2 (EG, CG / N = 60)									
0	0	0	0	9	15.00	31	51.67	20	33.33

In response to the first question 33.33% of the students strongly agreed, 46.67% agreed, 18.33% were neutral and none disagreed. In response to the second question 33.33% of the students strongly agreed, 51.67% agreed, 15.00% were neutral and none disagreed. Based on the answers to the first question, 81.67% of students found PE method more efficient than TE method, in terms of students' level of topic comprehension. The answers to the second question clearly showed that 85% of the students found the PE method more understandable than TE method. None of the 60 students found the TE method more efficient or understandable than PE method.

5. Conclusion and Discussion

The findings presented in Table 1 regarding the students' achievements in the pretest reflected that the students in EG and CG had almost equal levels of knowledge of SSMR before the experimental lectures took place.

Regarding students' knowledge level of SSMR after the experimental lectures, the findings in Table 2 revealed than the mean score of the students in EG was 21.79% higher than the mean score of the students in CG. This significant difference in *favor of the experimental group* showed that the pedagogical explanation method of teaching SSMR was more efficient than the traditional explanation method, in terms of students' knowledge level of SSMR.

The findings in Table 3 regarding the students' application level of SSMR, presented an even higher difference of 29.29% between the mean scores of the students in EG and CG, in *favor of the experimental group*. The students had to solve another similar matrix programming operation – sorting the numbers of a specified matrix row. The students in EG were much more successful than the students in CG in solving this problem because they applied the pedagogical steps similar to the ones they learned for SSMR operation. This significant difference in favor of EG showed that our pedagogical explanation method of teaching SSMR was especially more efficient than the traditional explanation method, in terms of students' application level of SSMR.

As Table 4 indicated, the mean overall score of the students in EG was 25.36% higher the than mean overall score of the students in CG. This significant difference in *favor of the experimental group* showed that our pedagogical explanation method was more efficient than the traditional explanation method in terms of the overall students' achievements in the posttest.

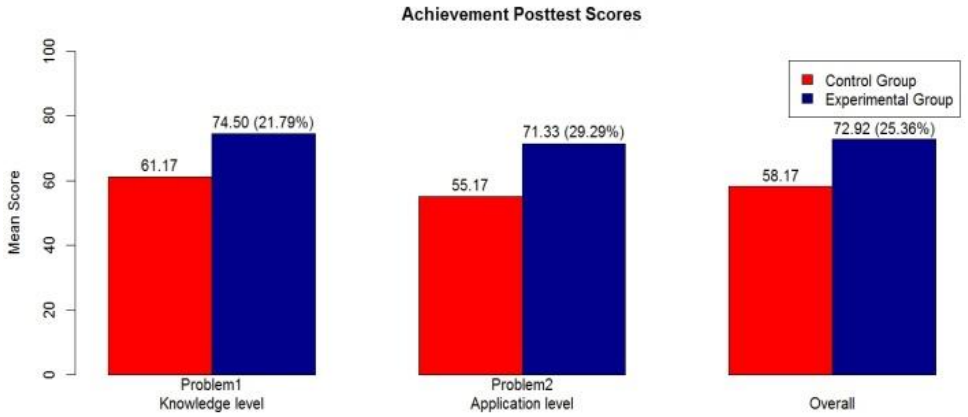

Figure 1. Achievement Posttest Scores

Finally, students' feedback on both explanation methods presented in Table 5, clearly reflected that majority of students (85%) considered the pedagogical explanation method of SSMR more efficient and understandable than the traditional explanation method. None of the students found TE method more efficient or understandable than PE method.

These results showed the effectiveness of teaching SSMR by using a pedagogical explanation method. This method positively influenced students' level of topic comprehension, which consequently improved their achievements. In order for students to understand better the other matrix programming operations, similar pedagogical explanation methods should be developed and used in IPCs. These methods should also be based on two important pedagogical techniques that we used in our PE method of teaching SSMR:

- *Visual tracing of MPOs by using concrete input values and specifiying matrix cell references for all posibble scenarios*. Visualizing the MPO from the pedagogical point of view is mandatory considering the positive effects of visualization in topics comprehension. It can be done by using various software tools ranging from standard office applications (presentation applications, word processors), programming languages to specialized graphics applications.
- *Inductive derivation of matrix functions from the corresponding array functions*. Majority of students in IPCs have not yet developed the "programming logic" to understand the traditional straitforward explanation of matrix programming operations. Consequently, higher level of comprehension can be accomplished by deriving the matrix operations from the corresponding array operations.

On a more general level, the results of this research suggested that pedagogical explanation methods should be developed and used for other topics in IPCs that students usually find difficult to understand, like recursion, linked lists, etc. Using these methods can be a very important factor in significantly increasing students' success in introductory programming courses.

References

[1] Bennedsen, J., & Caspersen, M. E. (2007). Failure Rates in Introductory Programming. *ACM Special Interest Group on Computer Science Education Bulletin*, 39 (2), 32-36.

[2] Brown, C. (2006). An Active Learning Pedagogy in a Programming Course. *Issues in Information Systems*, 2 (1), 124-128.

[3] Chetty J., & Barlow-Jones, G. (2014). Novice Students and Computer Programming: Toward Constructivist Pedagogy. *Mediterranean Journal of Social Sciences*, 5 (14), 240-251.

[4] Fares, S.C., & Fares, M.A. (2014). A Redesigned Pedagogy in Introductory Programming Reduces Failure and Withdrawal Rates by Half. World Academy of Science, Engineering and Technology. *International Journal of Social, Management, Economics and Business Engineering*, 8 (5), 1247-1250.

[5] Hadjerrouit, S. (2008). Towards a Blended Learning Model for Teaching and Learning Computer Programming: A case Study. *Informatics in Education*, 7 (2), 181-210.

[6] Mason, R., Cooper, G., & de Raad, M. (2012). Trends in Introductory Programming Courses in Australian Universities – Languages, Environments and Pedagogy. *Proceedings of the Fourteenth Australasian Computing Education Conference*, Melbourne, Australia, 33-42.

[7] Meyer, B. (2003). The Outside-In Method of Teaching Introductory Programming. *Proceedings of Symposium on Object Orientation* (IPSJ-SIGSE), Tokyo.

[8] Porter, L., & Simon, B. (2013). Retaining Nearly One-Third more Majors with a Trio of Instructional Best Practices in CS1. '*SISCSE 13 Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, 165-170.

[9] R Development Core Team (2012). R: A language and environment for statistical computing. Vienna: R Foundation for Statistical Computing, http://www.R-project.org.

[10] Robins, A. (2010). Learning edge momentum: A new account of outcomes in CS1. *Computer Science Education*, 20 (1), 37-71.

[11] Saeli, M., Perrenet, J., Jochems, Wim M.G., & Zwaneveld, B. (2011). Teaching Programming in Secondary School: A Pedagogical Content Knowledge Perspective. *Informatics in Education*, 10 (1), 73-88.

[12] Stephen, M., Elizabeth, A., Ogao, P., Franklin, W., & Ikoha, A. (2012). Choosing Tools of Pedagogy (Case of Program Visualization). *International Journal of Application or Innovation in Engineering & Management*, 1 (4), 35-40.

[13] Vitkute-Adžgauskiene, D., & Vidžiunas, A. (2012). Problems in Choosing Tools and Methods for Teaching Programming. *Informatics in Education*, 11 (2), 271-282.

[14] Vujosevic-Janicic M., & Tosic, D. (2008). The role of Programming Paradigms in the First Programming Courses. *Teaching of Mathematics*, 11 (2), 63-83.

[15] Woszczynski, A., Haddad, H., & Zgambo, A. (2005). Towards a Model of Student Success in Programming Courses. *ACM-SE Proceedings of the 43rd Annual Southeast Regional Conference*, 1, 301-304.

[16] Zingaro, D., Bailey Lee, C. & Porter, L. (2013). Peer Instruction in Computing: the Role of Reading Quizzes. *SIGCSE '13 Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, 47-52.