Roles of variables from the perspective of computer science educators

Roles of Variables as Seen by CS Educators

Mordechai Ben-Ari
Weizmann Institute of Science
Department of Science Teaching
Rehovot 76100
Israel
moti.ben-ari@weizmann.ac.il

Jorma Sajaniemi
University of Joensuu
Department of Computer Science
P.O.Box 111, FIN-80101 Joensuu
Finland
jorma.sajaniemi@joensuu.fi

ABSTRACT

Roles can be assigned to occurrences of variables in programs according to a small number of patterns of use that are both language- and algorithm-independent. Preliminary studies on explicitly teaching roles of variables to novice students have shown that roles are an excellent pedagogical tool for clarifying the structure and meaning of programs. This paper describes the results of an investigation designed to test the understandability and acceptability of the role concept and of the individual roles as seen by computer science educators. The investigation consisted of a short tutorial on roles, a brief training session on assigning roles to variables, a test evaluating the subjects' ability to assign roles, and a set of open questions concerning their opinions of roles. Roles were identified with 85 % accuracy, and in typical uses of variables with 93 % accuracy.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education—*computer science education*; D.m [**Software**]: Miscellaneous—*Software psychology*

General Terms

Human factors

1. INTRODUCTION

The primary method of teaching programming is to present examples so that the students can generalize from the examples to general principles of problem solving, and it is often worthwhile to formalize this process of generalization, and to explicitly teach program design techniques. The concept of *roles of variables* can be considered as another pedagogical technique within this tradition. In programming, variables are not used in an ad hoc way; instead, there are a *few* patterns that can describe almost all the uses of variables. Variable roles are a concept that is different from the algorithmic patterns that are frequently used as pedagogical aids, because the concept of roles focuses on the data flow through single variables. For example, in the long list of patterns given in [10], we find Pattern D1 (The Counter Pattern Using a Loop) with the following structure (in C++):

```
while (cost!=0) {loopCount++; ...}
```

and Pattern F1 (Performing an Action on Each Element of an Array) with the following totally different structure:

```
for (int index=0; index<MAX_ELEMENTS; index++) ...
```

In terms of roles of variables, however, both `loopCount` and `index` take on a predictable sequence of values and are assigned the same role (*stepper*); the choice of role is easily made just by examining the data flow. We believe that classroom discussions based upon roles of variables can contribute to the ability of students to understand and write programs.

The concept of roles of variables is based upon earlier work on variable use: Ehrlich and Soloway [4] and Rist [7] were interested in the mental representations of variables, while Green and Cornah [5] wanted to help maintenance programmers by providing a tool that would explain the behavior of variables. Our approach to the role concept is to find a comprehensive, yet small, set of characterizations for variables primarily for use in teaching.

Roles of variables were identified by Sajaniemi [8] who analyzed programs in several introductory Pascal textbooks. He found nine roles that covered 99 % of variables in novice-level programs. Later, Kuittinen and Sajaniemi [6] conducted an experiment during an introductory programming course, which compared traditional teaching with teaching that used roles and role-based animation. The results of this experiment indicated that the introduction of roles improves program comprehension and program writing skills.

In this paper, we are interested in computer science (CS) educators' attitude to the role concept and individual roles: if CS educators do not find the role concept intuitive and easy to apply, it would be unrealistic to expect them to use roles in teaching. To reveal how CS educators react to the new concept, we conducted an investigation that was designed to test the understandability and acceptability of the role concept and of the individual roles as seen by CS educators. Section 2 presents the concept of roles of variables, the investigation is described in Section 3 and the results are discussed in Section 4.

ITiCSE'04, June 28–30, 2004, Leeds, United Kingdom.

Table 1: Definition of the Roles in the Investigation

Fixed value	A variable that is initialized without any calculation and whose value does not change thereafter.
Stepper	A variable stepping through a succession of values that can be predicted as soon as the succession starts.
Most-recent holder	A variable holding the latest value encountered in going through a succession of values.
Most-wanted holder	A variable holding the "best" value encountered so far in going through a succession of values. There are no restrictions on how to measure the "goodness" of a value.
Gatherer	A variable accumulating the effect of individual values in going through a succession of values.
Transformation	A variable that always gets its new value from the same calculation from the values of one or more other variables.
Follower	A variable that gets its values by following the values of another variable.

2. THE ROLE CONCEPT

The role of a variable is defined according to its *dynamic* character as embodied by the succession of values the variable obtains and how the new values relate to other variables. For example, consider a variable used to store the latest input value given by the user. There is no possibility for the programmer to guess what values the user will enter; the role of such a variable is a *most-recent holder*. On the other hand, the sequence of values of a variable used as an index to iterate through an array is totally predictable as soon as the sequence starts; this variable is a *stepper*.

The original definition given in [8] contained nine roles, as well as a role *others*, for a few cases that could not be fit into the other categories. The list of roles was obtained by analyzing all the programs in three introductory textbooks. Later, as a by-product of the present study, a new role (*transformation*) was identified and added to the role set. The definitions of the roles as used in this research is given in Table 1. The *Roles of Variables Home Page* at `http://cs.joensuu.fi/~saja/var_roles/` contains fuller explanations and examples.

The concept of roles of variables is concerned with the *deep structure* [3] of the program: Does the variable hold a predetermined sequence of values, for example, the values of the index of a for-loop? Or does it hold the best value encountered so far, for example, when searching for the largest value in an array? The *surface structure* of the program, primarily its syntactic structure, is much less relevant to the concept of roles. The name of the variable, the places where it occurs within an expression and the relation between the expression and the enclosing assignment and control statements are not important in assigning roles.

Even though roles have technical definitions, they are a cognitive concept. For example, consider a variable that takes on the values of the Fibonacci sequence by adding up pairs of previous values in the sequence. A mathematician can predict the sequence as clearly as a novice can predict the sequence of values of the index of a simple for-loop, so she may assign the role of *stepper*, because the values "can be predicted as soon as the succession starts." On the other hand, a novice who has never seen the Fibonacci sequence before may assign the role of *gatherer*, because the variable accumulates the previous values.

3. THE INVESTIGATION

We conducted an investigation designed to study the understandability and acceptability of the role concept and of the individual roles as seen by CS educators. (Further details of the research methodology and results are available in an expanded version of this article [2].)

3.1 Methodology

The research materials consisted of web pages divided into three phases. The *tutorial phase* introduced the concept of roles of variables, followed by a section for each role containing: (a) the definition of the role (as given in Table 1), (b) a full sample program demonstrating the role, (c) additional examples of the use of the role, and (d) a list of additional properties that can assist in recognizing the role. Three roles (*one-way flag*, *temporary* and *organizer*) accounting for only 5.2 % of all variables in the analysis in [8] were not included in the tutorial in order to simplify and shorten it; the new role *transformation* was added as noted above. The tutorial consisted of a single web page yielding 8 pages when printed, as the subjects were encouraged to do.

Following the tutorial, subjects were presented with a *training phase*: this consisted of a sequence of six programs containing 24 variables taking on all of the roles described in the tutorial. After each program, subjects were given feedback on their assignments of a role to each variable The *analysis phase* was similar in format to the training phase; subjects were presented with six programs containing 24 variables, this time in a single web page. Upon assigning all of the roles, the results were sent by email to the authors.

```
program saw;
const last = 7;
type  ArrayType = array [1..last] of integer;
var   value: ArrayType;   { Values to be checked          }
      i:      integer;    { Index of array                }
      up: boolean;        { Current direction is up?      }
      ok: boolean;        { Does saw property still hold? }
begin
      writeln('Enter ', last, ' values:');
      for i:=1 to last do read(value[i]);
      up := value[1] < value[2];
      ok := value[1] <> value[2];
      i := 2;
      while ok and (i < last) do begin
          ok := (up and (value[i] > value[i+1])) or
                (not up and (value[i] < value[i+1]));
          up := not up;
          i := i + 1
      end;
      write('Values '); if not ok then write('do not ');
      writeln('form a saw.')
end.
```

Figure 1: Pascal program with controversial variables.

Table 2: Subjects' Selections for the Roles (percent)

Role	n	Role selected								
		FIX	STP	MRH	MWH	GTH	TRN	FOL	OTH	DNK
FIX	5	**91**		7		2				
STP	6		**91**	2		1	4	2		
MRH	2	7	1	**92**						
MWH	3		1	1	**79**	3	3	10	1	3
GTH	2	1	1	10	1	**60**	26	1		
TRN	3	9	1	7	3	1	**75**	4	1	
FOL	2			2				**96**		2
OWF	1			8	10	10	61	6	**6**	

While the roles in the training phase were straightforward, some of the variables in the analysis phase were "controversial," i.e., borderline cases. Such programs would not normally be shown to novices, but these variable usages were included in order to validate the definitions of the roles. One such program was an iterative program for constructing elements of the Fibonacci sequence.Another (Figure 1) checks if a sequence of values forms a "saw," in which the direction of change of the values alternates. The controversial variables are `up` and `ok`. The former is ostensibly a stepper, because we can predict that its values alternate between `true` and `false`, though other roles are plausible since the initial value of the variable is computed from the values of other variables. The variable `ok` is a *one-way flag*—one of the three roles not included in the tutorial. A *one-way flag* is the role assigned to a variable that may change its value only once; this role is frequently used for a variable of type boolean used as a "flag." We wanted to see if the absence of this role would be missed by the subjects and what choices they would make.

The first version of the material was pretested by using five CS educators as subjects. They used materials containing eight programs intended to form the analysis phase. The time needed to complete the task varied from 28 to 90 minutes with mode being 60 minutes. The final materials can be found at `http://cs.joensuu.fi/~saja/role_survey/`.

3.2 Sample Demographics

Fifty-three computer science educators volunteered to participate in the investigation. They were recruited by publicizing the URL containing the research material among CS educators in the authors' countries, as well as on mailing lists belonging to the special interest groups in computer science education and psychology of programming.

One subject selected a quit option while working on the material; another's result was discarded as apparently consisting of randomly selected answers. The results of the remaining 51 subjects were used in the analysis. Subjects represented both high-school teachers ($n = 8$) and university or college teachers ($n = 38$). Some subjects had been worked at both levels ($n = 6$) while some did not report teaching at either level ($n = 11$).

3.3 Results

Table 2 displays the selections made by the subjects for each role in the analysis phase. The column labeled with n gives the number of variables having the role given in the first column. Other columns are labeled with the possible roles that could have been assigned. Therefore, an entry (row, col) in the table gives the percentage of the occurrences when role *col* was assigned to a variable whose correct role was *row*. The diagonal, the percentage of correct assignments, is emphasized. OTH means that a subject thought the variable to have some *other* role not listed in the tutorial, while DNK means that a subject *did not know* which role should be used. OWF is the role *one-way flag*, discussed above.

Most roles are identified by at least 90% accuracy. The low success in identifying of *most-wanted holders* and *gatherers* can be explained by controversial variables. In non-controversial cases, *most-wanted holders* were identified correctly in 91% of the cases, and *gatherers* in 94% of the cases. Only 75% recognized *transformations*, though even here, in a simple case the role was recognized with 90% accuracy.

Excluding the *one-way flag*, subjects made on average 3.4 errors in the classification of the 23 variables (average accuracy of 85 %), and 1.3 errors in the classification of the 19 non-controversial ones (average accuracy of 93 %). Subjects making at most 1 error ($n = 10$) were more experienced both in teaching introductory programming courses (two-tailed t test, $t = 2.310, df, p = .0317$) and in teaching advanced CS courses ($t = 2.944, df, p = .0080$) than subjects making at least 5 errors ($n = 12$).

We also looked at the distribution of error selections for high and low performers. To obtain roughly the same amount of errors for both groups, high performers were defined to be subjects with at most 3 errors (making a total of 60 errors) and low performers were subjects having at least 6 errors (with a total of 63 errors). We counted the number of erroneously selected roles for each variable; for all variables (except the *one-way flag*) low performers selected a wider variety of roles. The difference is statistically significant (paired t test, $t = 3.943, df = 22, p = .0007$). For the *one-way flag* the difference is opposite: high performers made a wider variety of selections (5 different roles) than low performers (3 different roles).

3.4 Error analysis

Recall from Section 2 that roles of variables are used to describe the deep, rather than the surface, structure of the program. When analyzing errors this distinction is important: for example, an error that confuses two roles with different deep structures but similar surface structures reveals that the subject has a weak understanding of the distinction between two roles. This difference between surface and deep structures was explicitly mentioned by one of the subjects who was analyzing the role of the variable `second` that holds the second largest value seen so far in a search for the two largest values in a sequence of values:

> At the code [i.e., surface] level `second` seems to have two roles "follower," and "most-wanted", but semantically [i.e., at the deep level] it has just one: "most-wanted." It is the best choice for the criterion "to be the 2nd largest."

Errors in the assignment of roles were frequently caused by *atypical* use of the variables. For example, *steppers* are typically used for sequences whose values increase or decrease monotonically, often in an arithmetic progression. This typical use is so dominant that it may be hard to recognize that other successions, such as the alternating sequence of values {`true`, `false`, `true`, `false`, ...} assigned to the variable `up`, are just as predictable and therefore this variable should also

be assigned the role *stepper*. Many subjects erred, because although the surface structure is typical (`up:=not up` is syntactically similar to the typical `i:=i+1`), the deep structure with its nonmonotonic sequence is not. Being misled by an atypical use of a variable does not indicate that the role is counterintuitive; instead, it is a *pedagogical challenge* to learn to gloss over surface structure in order to analyze deep structure. The controversial variables presented in the investigation involved both atypical surface and atypical deep structures. The *most-wanted holder* `second` has an atypical surface structure, while the *stepper* `up` has an atypical deep structure.

Errors made by **high performers on non-controversial variables** are potential indicators that the entire concept is not viable. There were only two error types of this kind. Several subjects erred in assigning the role to an array whose elements were read in at the beginning of the program. Since each element is a *fixed value*, the array is a *fixed value*, but the subjects considered it to be a *most-recent holder*, because each component of the array holds the latest value read from the input. The error was less frequent on the subsequent occurrence of an array in a program, so we believe that with more experience, subjects would cease to make the error.

The second error concerned the role of *transformation* which identifies cases where a variable has no independent existence, but merely serves to contain a value obtained by computation, for example, a unit conversion from a number to a percentage, or from degrees to radians, or a split of a number into its quotient and remainder upon division by another number. The difference between a *transformation* and the role of the original variable, or between a *transformation* and some other role having the same surface structure, was not always apparent to the subjects. For example, the variable `factor` was a transformation computed as `percent/100`, but thereafter not modified. Clearly subjects were justified in assigning the role *fixed value* to that variable.

There were also two types of errors made by **high performers when assigning roles to controversial variables.** The first type appeared in three cases when an atypical surface structure misled some subjects to select the role whose typical surface structure matched the variable in question. For example, if the variable was assigned different values in the two alternatives of an if-statement, some subjects did not recognize that the variable was still assigned a value only once and thus should be assigned the role *fixed value*.

The second error type (one case) was caused when an atypical deep structure triggered a large variety in the roles suggested by the high performers. In this case (the variable `up` discussed above), the subjects searched for roles with a more appropriate deep structure. Since the roles are designed to characterize distinct deep structures, there were no other roles that were appropriate for this case resulting in a variety of answers.

In general, **low performers** made the same types of errors for both controversial and non-controversial variables as did high performers and made them more often. Other errors that they made can be explained by a tendency to make decisions based on surface structure only. The role *most-wanted holder* was assigned even when no possible "measure of the goodness of the value" existed; *gatherer* was assigned when no "accumulation" was being carried out; *follower* was assigned when it did not contain values of the variable being followed, etc.

An interesting exception to the similarity of errors is the case of the controversial variable with atypical deep structure (the variable `up`): high performers spread their errors evenly among many roles, while low performers concentrated their errors on a single role *transformation*, the one with closest surface structure. The same behavior was seen in the case of the *one-way flag*, which was the role that was missing from the tutorial. High performers assigned a larger variety of roles than did the low performers. When confronted with a variable where none of the roles seems to appropriate, low performers tend to look at the surface structures. In this case, the structure of the expression matched a few roles only, so the low performers made their selection among these. On the other hand, high performers looked at the deep structures, and, as discussed above, found that a variety of roles were plausible.

The roles *most-recent holder* and *transformation* were most often assigned in case of doubt, presumably, because their definitions are the least specific. Any variable holds the most recent value of some calculation, even though *most-recent holder* is reserved for "raw" data such as input values. Similarly, many variables are the result of computation from other variables, even though *transformation* is intended to be used in specific cases like scaling values.

4. DISCUSSION

The fact that subjects agreed with our assignment of the roles after such a short introduction is encouraging (more than 90% of the cases for non-controversial variables for every role except *transformation*). The results make it clear that increased teaching experience improves the ability to assign roles correctly, and they indicate that experts have little problem with the role concept, supporting the assumption that roles represent *tacit expert knowledge*. Roles are not hard for non-experienced teachers either, since for the 19 non-controversial variables, 86% of the subjects made at most two errors.

Many subjects stated in their comments that they had had problems in remembering the definitions of the roles or that the definitions were ambiguous, but the same subjects scored between zero to two errors on non-controversial variables, indicating that they understood, perhaps subconsciously, the deep structure of variables represented by the roles. Subjects' comments on the role concept in general were mostly positive, and they believed that roles could contribute to understanding programs.

The *one-way flag* role that was not included in the tutorial garnered the largest number of alternative suggestions for other roles. This provides evidence that the *one-way flag* is a distinct role, not subsumed by or similar to the others. New roles suggested by the subjects for this variable, including *checker*, *guardian*, *state* and *latch*, are consistent with our definition of the role. Furthermore, the dearth of suggestions for new roles for the other variables supports our claim that the role set is sufficient for the analysis of variables in novice-level programs.

The only role that caused frequent confusion was *transformation*. As discussed above, it is intended to identify cases where a variable has no independent existence, but merely serves to contain a value obtained by computation. In a

sense, this role "usurps" the role or roles assigned to the variables from which the transformation is computed. The definition of this role has to be clarified.

Variables become controversial if either the surface structure or the deep structure is atypical. The ability to recognize and go beyond an atypical surface structure is gained by increased expertise and developing this ability is the task of the teacher. Variability in surface structures is so large that it makes no sense to add or modify roles to take surface structure into account. On the other hand, atypical deep structure is a sign that a new role might be needed. This claim is justified by the similarity in the subjects' approach to the atypical deep structure and to the missing role. Since we want to keep the number of roles small so that they can be used in introductory teaching, we prefer that new roles that rarely occur should be embedded within the existing roles.

The tutorial and training materials were deliberately kept short to encourage compliance by our subjects. Obviously, the number of examples should be much greater in order to explore the ramifications of the definitions of the roles in a wider selection of programs. This will not be a problem in an educational setting, where the roles can be introduced gradually during an introductory course and reinforced in all the examples and exercises.

The error analysis suggested improvements in the tutorial. First, it should stress that roles concern the deep structure of variables, even though roles can often be identified from typical surface structures (for example, the index of a for-loop is almost certainly a *stepper*). Second, the tutorial has to give criteria for distinguishing *transformations* and *gatherers* from other "computationless" roles. Finally, the application of roles to structured data types must be covered thoroughly.

5. CONCLUSION

The concept of roles of variables can be used as a pedagogical technique to teach how the constructs of a programming language work together to implement the solution of a problem. Preliminary results of using roles in teaching elementary programming indicate that the introduction of roles improves program comprehension and program writing skills. In this paper, we were interested to find out how computer science educators react to this new concept and to the individual roles. The outcome of the investigation is encouraging because CS educators accepted the concept of roles as intuitive and found it easy to assign roles consistently.

Even in those cases where assignment is controversial, the debate itself can be an excellent pedagogical tool for clarifying the structure of programs in introductory courses. It is important to emphasize that we do not regard roles as an end in themselves and we do not think that students should be graded on their ability to assign roles. Roles of variables are design rules and pedagogical aids intended to help novices over the hurdle of learning programming.

Roles of variables can also change the way that program visualization and animation are carried out [9]. Traditional systems such as Jeliot [1] provide visualizations that operate on the programming language level; therefore, the representation and animation of variables is uniform reflecting the surface structure of the program, not its deep structure. Role-specific representation of variables and role-specific animation for operations should result in visualizations on a higher level that will be more informative to students.

Future research on roles will include: (a) cognitive studies to investigate if roles are truly part of the knowledge structure of experts, though even if the answer is negative, it would not rule out their pedagogical use; (b) further empirical research in classrooms in order to determine if roles are pedagogically useful; (c) development and evaluation of program animation for visualization of roles; (d) extension of the role set to cover other programming paradigms and other expertise levels.

6. ACKNOWLEDGMENTS

We would like to thank all those who volunteered to participate in the study, and especially Noa Ragonis for her extensive comments on the first version of the tutorial and on a draft of this paper.

7. REFERENCES

[1] M. Ben-Ari, N. Myller, E. Sutinen, and J. Tarhio. Perspectives on program animation with Jeliot. In *Software Visualization: International Seminar*, Lecture Notes in Computer Science 2269, pages 31–45, Dagstuhl Castle, Germany, 2002.

[2] M. Ben-Ari and J. Sajaniemi. Roles of variables as seen by CS educators. Technical Report A-2003-6, University of Joensuu, 2003. `ftp://ftp.cs.joensuu.fi/pub/Reports/A-2003-6.pdf`.

[3] F. Détienne. *Software Design – Cognitive Aspects*. Springer Verlag, London, 2002.

[4] K. Ehrlich and E. Soloway. An empirical investigation of the tacit plan knowledge in programming. In J. C. Thomas and M. L. Schneider, editors, *Human Factors in Computer Systems*, pages 113–133, Norwood, NJ, 1984. Ablex Publishing Co.

[5] T. R. G. Green and A. J. Cornah. The programmer's torch. In *Human-Computer Interaction — INTERACT'84*, pages 397–402. IFIP, Elsevier Science Publishers (North-Holland), 1985.

[6] M. Kuittinen and J. Sajaniemi. First results of an experiment on using roles of variables in teaching. In *EASE and PPIG 2003, Papers from the Joint Conference at Keele University*, pages 347–357, 2003.

[7] R. S. Rist. Knowledge creation and retrieval in program design: A comparison of novice and intermediate student programmers. *Human-Computer Interaction*, 6:1–46, 1991.

[8] J. Sajaniemi. An empirical analysis of roles of variables in novice-level procedural programs. In *Proceedings of IEEE 2002 Symposia on Human Centric Computing Languages and Environments (HCC'02)*, pages 37–39. IEEE Computer Society, 2002.

[9] J. Sajaniemi. Visualizing roles of variables to novice programmers. In J. Kuljis, L. Baldwin, and R. Scoble, editors, *Proceedings of the Fourteenth Annual Workshop of the Psychology of Programming Interest Group (PPIG 2002)*, pages 111–127, 2002.

[10] C. Sollohub. C++ in Hypertext. `http://cs.nmhu.edu/personal/curtis/cs1htmlfiles/Cs1text.htm`, 2001.