Roles of Variables in Object-Oriented Programming

Pauli Byckling
Department of Computer Science
University of Joensuu
Joensuu, Finland
pbyckli@cs.joensuu.fi

Petri Gerdt
Department of Computer Science
University of Joensuu
Joensuu, Finland
pgerdt@cs.joensuu.fi

Jorma Sajaniemi
Department of Computer Science
University of Joensuu
Joensuu, Finland
saja@cs.joensuu.fi

ABSTRACT

Roles can be assigned to occurrences of variables in programs according to a small number of stereotypical patterns of use. Studies on explicitly teaching roles to novices learning procedural programming have shown that roles are an excellent pedagogical tool for clarifying the structure and meaning of programs and that their use improves students' programming skills. This paper describes the results of an investigation designed to test the understandability and acceptability of the role concept and of the individual roles in novice-level object-oriented programming.

The role set used in procedural programming was found to be suitable for describing variable and attribute behavior in object-oriented programming but the need for some supplementary roles—due to the early introduction of linked structures in object-oriented programming—was also identified. CS educators had little problems with identifying roles in typical uses of variables. Every role was identified in typical uses of variables by 70–100 % accuracy. Subjects' comments on the role concept were mostly positive.

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, Theory

Keywords

Roles of variables, Object-oriented programming, Computer science education

1. INTRODUCTION

Teaching programming involves more than just teaching the syntax and semantics of a programming language. The student must learn how the constructs of the language work together to implement the solution of a problem. 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* [9, 8] 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* stereotypical behaviors of variables. Sajaniemi [9] analyzed programs in several introductory Pascal textbooks with an intention to find a comprehensive, yet small, set of characterizations for variables. The search resulted in nine roles that covered 99 % of variables—a small enough set to be covered in programming teaching. Later, one more role was added to the set [2]. Kuittinen and Sajaniemi [10] conducted a classroom experiment during an introductory programming course comparing traditional teaching with teaching that used roles and role-based animation. The results indicate that the introduction of roles considerably improves program comprehension and program writing skills [3, 10]. Roles can be considered to be extremely simple patterns that capture the basic forms of data element behavior. Thus roles occur in practically all programs and they form an essential part of experts' programming knowledge [11].

The role concept is applicable to other programming paradigms with small adjustments. Previously its applicability to functional programming has been studied [6]; this paper examines its application to object-oriented programming. We analyzed programs in two introductory Java programming books in order to find roles needed in object-oriented programs, and conducted an investigation methodologically similar to a previous one by Ben-Ari and Sajaniemi [2]. The rest of this paper is organized as follows. Section 2 describes the role concept: Section 3 presents the investigation and its results followed by discussion in Section 4. Section 5 contains the conclusion.

2. THE ROLE CONCEPT

The role of a variable is defined according to the dynamic character of the variable embodied by the succession of values it obtains, i.e., its behavior. For example, in the role of a *stepper*, a variable is assigned a succession of values that is known in advance as soon as the succession starts. The role concept does *not* concern the way a variable is used in the program. Informal definitions of the roles are given in Table 1. The *Roles of Variables Home Page* at [8] contains fuller explanations and examples.

Roles are concerned with the *deep structure* [5] of the program: Does the variable 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

OOPSLA'05, October 16–20, 2005, San Diego, California, USA.
ACM 1-59593-193-7/05/0010.

Table 1: Informal role definitions.

Role	Informal definition
Fixed value	A variable which is initialized without any calculation and whose value does not change thereafter.
Stepper	A variable stepping through 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 value(s) of other variable(s).
Follower	A variable that gets its values by following another variable.
One-way flag	A two-valued variable that cannot get its initial value once its value has been changed.
Temporary	A variable holding some value for a very short time only.
Organizer	An array which is only used for rearranging its elements after initialization.
Other	Any other variable.

syntactic structure, is much less relevant to the roles.

In object-oriented programming, roles concern attributes in classes and variables inside methods. Moreover, some objects can also have a "variable-like" behavior. Two special cases appear: In Java, e.g., a single String-type or an Integer-type object capsulates just one value. All the events directed to the object will have an effect on the value of the only attribute and the whole object behaves just like a primitive variable. In such cases, the object is considered to have the role of its only attribute. In the following, the term "variable" will be used to denote elements that can be assigned a role, whether they are actually variables, attributes, or single attribute objects.

Usually object references are used to call the methods of the objects, or to reference their variables. However, in some cases these references behave like variables and are therefore given roles. In such cases the role describes the reference, not the object it refers to. For example, in Java a reference to an Object (the superclass of all classes) can hold the current item obtained from a data structure during an iteration through its values; hence it can be given a role. Pointers are treated similarly to object references: the role of a pointer describes how the pointer behaves and not how the pointed variable or object behaves.

In larger programs, the role of a variable may change during its lifetime. The most special instances of role changes can be found among attributes of inherited classes. Thus it is possible that a variable declared in the parent class can behave differently in two different inherited classes. These examples must, however, be considered as special cases and assumingly they are rare in real life programs.

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.

To see if the roles suggested for novice-level procedural programming are sufficient for novice-level object-oriented programming, we analyzed all programs in two introductory Java programming textbooks [7, 12]. Two researchers assigned independently roles to all variables in the first textbook. They then discussed variables and their roles one by one, recorded differences in role assignments and resolved contradicting interpretations. The same procedure was then repeated with the other textbook.

Table 2: Proportion of roles in two introductory Java programming textbooks.

Role	[7]	[12]
Fixed value	67	44
Stepper	14	21
Most-recent holder	3	9
Most-wanted holder	1	0
Gatherer	1	2
Transformation	8	15
Follower	1	1
One-way flag	3	3
Temporary	0	1
Organizer	0	1
Other	1	4
Total	100	100

There were 92 programs in the first book and 107 programs in the second book. The programs contained 948 variables out of which 98 % were assigned a role. The remaining 16 variables were classified as *other*; 10 of these were arrays. The discussions revealed a few cases—especially in the context of data structure manipulation—where the roles did not exactly fit the behavior of variables but no new roles were introduced. Table 2 presents the proportions of roles in the textbooks. The most distinctive detail is the large amount of *fixed values*, that can be explained by the amount of parameters of methods which all were *fixed values*. The parameter subcategory covers 44 % of all the *fixed values* in the first book and 43 % in the second book.

3. INVESTIGATION

In order to validate the definitions of the roles in object-oriented programming from the perspective of computer sci-

ence educators we conducted a web-based investigation. In a similar study by Ben-Ari and Sajaniemi [2] CS educators found the concept intuitive and easy to apply with novice-level procedural programs.

3.1 Methodology

The investigation was conducted on the Internet. 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 applying the role. The programs were written in Java, but the concept of roles of variables is not language-dependent so other object-oriented languages could have been used. For brevity, full technical definitions of the roles were omitted; instead, the presentation was closer to the way roles could be introduced to novices learning to program.

Variables in the investigation were limited to scalar variables, object references and parameters, though the roles can be applied to more complex types such as pointers and records. All programs were also designed so that *role changes*, where the same variable takes on multiple roles in succession, did not occur. Similarly to the previous study [2], three roles (*one-way flag*, *temporary* and *organizer*) accounting for only 4 % of all variables were not included in the tutorial. The tutorial consisted of a single web page yielding 8 pages when printed, as the subjects were encouraged to do to improve readability.

Following the tutorial, subjects were presented with a *training phase*: this consisted of a sequence of six programs containing 27 variables taking on all of the roles described in the tutorial. The subjects assigned roles using radio buttons; to reduce the demands on short-term memory and to ensure accuracy in the use of the roles, each button label was linked to the corresponding role definition. After each program, the subject was given feedback on his or her assignment of a role to each variable.

The investigation was concluded with the *analysis phase* where subjects assigned roles to variables in a large, fully functional program, which implemented a "person membership register". The program had 607 lines (comments and empty lines included) and comprised of 7 classes. Two of the classes inherited a common superclass, one was an inner class and one class was an abstract class. In the previous investigation [2] the *analysis phase* was similar in format to the training phase having six small novice-level programs containing 24 variables in a single web page. In order to use object-oriented programming features, e.g., class hierarchies, one larger program was used in our investigation instead.

We extracted six code fragments from the program for the analysis phase. Although the program as whole was more complex than typical introductory programs, the code fragments needed in the tasks reflected novice-level programs. From these fragments we chose 23 variables to be analyzed. The web page in the analysis phase was divided vertically into two frames; the left frame contained all six tasks and the right frame presented the necessary code fragment when some of the variable names in a task was clicked. In order to make the comprehension task easier, occurrences of a variable were highlighted when a subject clicked on its name in the task frame. The opportunity to examine the program as whole was also given for the subjects, but this was advised to be left as an issue of personal interest only.

Table 3: Occurrences of the roles in the analysis phase programs.

Role	Program						Total
	1	2	3	4	5	6	
Fixed value	1	3			1	1	6
Stepper	1	1		1	3		6
Most-recent holder			2		1		3
Most-wanted holder				1			1
Gatherer		1	1				2
Transformation				1		2	3
Follower	1					1	2

Subjects were given the option to suggest a new role by themselves or to indicate that they did not know which role to select, although they were encouraged not to do so. They also had the opportunity to append comments to their choices. If roles were not selected for all variables, a list of omitted variables was provided, and the subject could return to the original page to finish the task. After assigning roles, subjects were asked for general comments on the role concept or specific roles. Moreover, they were asked to indicate their length of experience teaching introductory programming and/or advanced CS courses in high school and/or college or university. We asked for names and email addresses so that we could send them copies of the research results, though subjects were allowed to remain anonymous if they so wished. The results were sent by email to the authors at the end of the analysis phase. Subjects were also allowed to click on a "quit" link following the tutorial and the training phase.

Subjects worked remotely on the web-based material at their own pace. They were instructed to start with the tutorial and training phase, and only then to perform the analysis phase. The initial page contained direct links to the various phases, so that there was no need to complete the whole task in a single session. The material was pretested with three programmers and some minor corrections were made to the program code. Table 3 summarizes occurrences of roles in the analysis programs. The final materials can be found at http://cs.joensuu.fi/~saja/oo-role-survey/.

3.2 Sample Demographics

Subjects were recruited by publicizing the URL containing the research material on mailing lists belonging to special interest groups in computer science education, object-oriented programming and psychology of programming. In addition to this, the e-mail was sent personally to 370 programming teachers whose addresses were collected from Internet. Fourty-four computer science educators volunteered to participate in the investigation. One subject selected a quit option due to lack of time. The results of the remaining 43 subjects were used in the analysis.

Subjects represented mostly university or college teachers ($n = 34$). Some subjects had been worked also as high-school teachers ($n = 3$) while some did not report teaching at either level ($n = 9$). Figure 1 summarizes subjects'

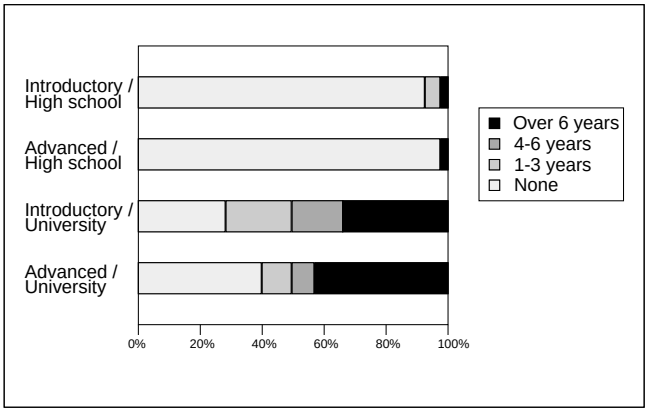

Figure 1: Subjects' teaching background.

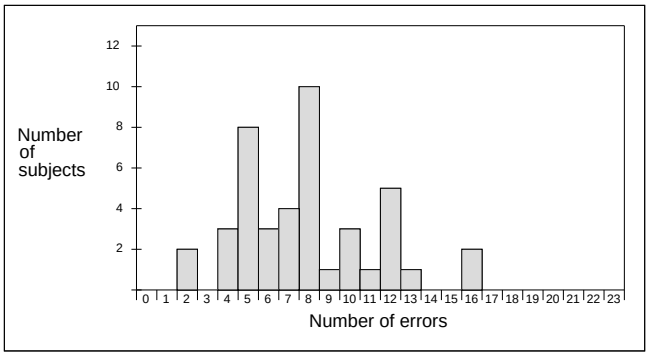

Figure 2: Frequencies of error scores.

experience in teaching either introductory programming or advanced CS courses in either high-school or at higher level.

3.3 Results

Table 4 displays the selections made by the subjects for each role in the analysis phase. For each of the roles in the analysis programs, there is a row in the table labelled by that role, together with n, the number of variables having that role. The columns are labelled 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. For example, the first row describes subjects' selection for the six *fixed values* in the analysis programs. In 71 % of cases, subjects' selections were correct, while in 9 % of cases *most-recent holder* was selected etc. 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.

Most roles are identified with 60–71 % accuracy. Figure 2 contains a frequency graph depicting the distribution of subjects' error scores. There were an average of 7.8 errors in the classification of the 23 variables (i.e., average accuracy of 62 %). Subjects having longer teaching experience made in general less errors than less experienced subjects but the difference was not statistically significant.

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 7 errors (making a total of 102 errors) and low performers were subjects having at least 12 errors (with a total of 105 errors). We counted the number of erroneously selected roles for each variable; low performers selected a wider variety of roles and the difference was statistically significant (paired t test, $t = 6.200$, $df = 22$, $p < .0001$).

4. DISCUSSION

The overall accuracy of assigning roles to variables, 62 %, was much lower than in the earlier study concerning roles in procedural programming [2] where the overall accuracy was 85 % and excluding "controversial" cases, i.e., variables representing borderline cases and having several interpretations depending on the point of view, even 93 %. However, the present study differs from the previous in several respects. First, the program was much longer and more complicated; variables were updated with assignments dispersed farther away from each other; and control flow was harder to comprehend because of nested method invocations and smaller indentation step. These choices were made in order to obtain better object-orientation but they made comprehension of the behavior of variables harder. Second, the present study contained more controversial cases than the previous study because the acceptability of normal cases had already been validated by the previous study. Thus, the lower accuracy in the present study does not mean that roles would be of lesser importance in object-oriented programming but that the tasks were harder than in the previous study. More important than looking at the error rates is to analyze various error types and discuss their meaning to the role concept in a general level and especially in object-oriented programming. The following subsections concentrate on this analysis.

4.1 Errors Made by High Performers

In overall, high performers concentrated their errors on a few variables while low performers spread their errors evenly among many variables. Moreover, high performers were more definite in their classifications and their errors were explicable which confirms the results of the previous investigation [2]. Errors made by high performers are potential indicators that the entire role concept or the definitions of individual roles are not viable, and therefore deserve to be studied carefully.

Errors related to individual roles: Errors made by the high performers can be divided into two categories depending on whether they are related to some certain roles or were affected by some other factors. Three error types were related to the individual roles. First, roles are *cognitive concepts* whose borders are not strict and may vary from person to person. Two variables fell into this category: a *fixed value* that was set by searching an appropriate value in an array, and the left index in a binary search. The latter variable resulted in the most even distribution of suggestions over all roles among high performers, and was the only variable with a larger selection of wrong role suggestions among high performers than among low performers. The high variability among high performers suggests [2] that the variable does not fit any of the roles well.

Second, the *transformation role* caused confusion just like in the previous study [2]. A *transformation* identifies cases

Table 4: Subjects' selections for the roles (percent).

Role	n	Role selected									Total
		FIX	STP	MRH	MWH	GAT	TRN	FOL	OTH	DNK	
FIX	6	**71**	3	9	2		10	2	1	3	100
STP	6		**64**	18	5	9		2	1	1	100
MRH	3	5	5	**64**	5		16	2	2	1	100
MWH	1	1	1	4	**54**		37	3	1		100
GAT	2	12	1	1	3	**60**	19	3	3	1	100
TRN	3	3		19	2	4	**67**	3	4	1	100
FOL	2	1	16	16	4		5	**57**	2	1	100

where a variable has no independent existence, but merely serves to contain a value obtained by computation, for example, a unit conversion. 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, especially as the tutorial did not contain examples of all surface structures occurring in the analysis phase. This problem was evident with three variables.

Third, *atypical* use of a role mislead subjects in two cases. For example, *steppers* are typically used for sequences whose values increase or decrease monotonically, often in an arithmetic progression. In the most typical case the role *stepper* was recognized with 100 % accuracy. This typical use is so dominant that it may be hard to recognize other occasions of a *stepper*, such as stepping through the elements of a linked list. Moreover, the mental model of typical structures is strong with experts; e.g., in object-oriented programming the concept of an iterator is so obviously existent, that it may cause ambiguity to the role *stepper*. This is clearly stated by one of the subjects:

> S23: There would seem to be a role missing between *stepper* and *most-recent-holder* in the case of an iteration over a linked data structure; i.e., the 'iterator' role. I have not classified these as *stepper*, as that seems to want to be a numeric value. The iterator role is an abstraction of *stepper*.

The other variable expressing atypical use was a *most-recent-holder* which did not obtain its new values as input—which is the typical case—but from the elements of an array.

Errors not related to individual roles: There were three error types that did not concern the behavior of a variable, i.e., not directly related to the individual roles. As mentioned earlier, roles are cognitive rather than technical concepts. This means that *other parts of the mental model* constructed by a program comprehender may affect his or her interpretation of the role. Understanding a variable, e.g., through its utilization or responsibilities can lead to an intelligible but defective interpretation of the role. Such interpretations are, however, based on some other element than the succession of values, which is the basis of the role concept. Errors in two *steppers* can be explained in this way. One *stepper* generated new unique id:s from the sequence 1, 2, ... and it was often assigned the role of *most-recent holder*; the other counted the number of changes and it was often assigned the role of *gatherer*.

Second, even though a role describes the behavior of a variable in an individual object, the *set of all instances of an attribute* seemed to affect role assignment in a case where the role *fixed value* was sometimes confused with *most-recent holder*. Even though the value of the attribute did not change within a single object, new instances of the attribute in new objects may have given the impression of a value succession. This error type is special to object-oriented programming but a similar effect has been found with arrays in procedural programming [2].

Third, with one variable a form of *set effect* [1, p. 268] was found. The surface structure and name of the variable were similar to those of the previous variable in the task, and many subjects assigned the role of the first variable to the latter variable, also. This error has no relation to the roles but is raised by the form of the experimental materials.

4.2 Suitability of the Role Set

In general, the role set used in the investigation seems to be largely suitable for its purpose. Discarding controversial variables and the variable with the set effect, the average accuracy of assigning roles is 83 %. The naturalness of the roles is further supported by the decrease in the number of different error types with increasing experience. Most importantly, uncertainty decreases distinctively when experience increases. The number of "I don't know" answers is zero and the number of "some other role" answers is minimal for the high performers. Moreover, the erroneous interpretations are highly similar within this group.

The 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 [2].

Variables used for traversing data structures seem not to fit the existing roles well. The current investigation raised the need for a new role "traverser" for variables that go through data structures. In object-oriented programming this kind of role is typically called iterator. Other roles needed for data structures might be "container" describing the whole structure, and "handle" that gives access to some certain element of a data structure, e.g., its first or last element. However, the real need for these roles must be analyzed before taking them into use.

In the current role set, "traverser" is a special case of *steppers*—the second most frequent role. Another frequent error with *steppers* concerned a counter variable that was assigned the role of *gatherer*. The granularity of perceiving different behaviors as similar varies among professional pro-

grammers [11] and the above findings suggest that the role *stepper* could be divided into several more specific roles.

4.3 Roles and Object-Oriented Programming

In novice-level *procedural* programs, current roles apply to 98 % of all variables [9]. However, our investigation suggests that novice-level *object-oriented* programs are considerably more complicated than novice-level procedural programs and that some new roles are needed. Deficiencies in the role set were first detected as ambiguous cases in analyzing variables in the textbooks and they were reinforced by the web-based investigation. The new roles concern data structures and will probably be needed in procedural programming, also, but only at the intermediate level. On the other hand, all the present roles apply to object-oriented programming and were identified in object-oriented textbooks. Thus there is no need to alter present roles for the purposes of object-orientation.

In larger programs it can be hard to extract the behavior of a variable when the lines affecting its value are dispersed. In object-oriented programming, data flow is decentralized and all lines affecting a variable are not necessarily in the same method—in fact they can be in different classes—and control flow is harder to reveal than in procedural programming [4]. One subject mentioned this problem explicitly:

> S26: It seemed that it was harder to determine roles for the variables which were used in more methods than one.

This does not, however, mean that roles would be less important in object-oriented programming. On the contrary, explicit role information in the form of comments written by the author of a program might help expert programmers in program comprehension; knowing that a variable is, e.g., a *stepper* indicates the succession of values it may obtain.

Finally, some subjects pointed out in their comments that they had problems in embedding the role concept into object-oriented thinking. This may be due to the fact that expert object-oriented programmers are used to analyze programs in terms of patterns that represent stereotypical class hierarchies and work at a higher abstraction level than variable roles. On the other hand, many subjects reported that the role concept was natural and provided a new way of thinking about variables. Thus the role concept seems to fit object-oriented programming well and problems in relating it to patterns are supposedly rare if roles are introduced in the first programming courses.

5. CONCLUSION

We have studied the applicability of the concept of variable roles in the context of object-oriented programming by classifying all variables in two novice-level object-oriented programming textbooks and by testing the understandability and acceptability of the role concept as seen by CS educators. The role set used in procedural programming was found to be suitable for describing variable and attribute behavior in object-oriented programming but the need for some supplementary roles—due to the early introduction of linked data structures in object-oriented programming—was also identified.

CS educators had little problems with identifying roles in typical uses of variables. Moreover, errors caused by problems in understanding how a program works suggest that explicit role information can facilitate program comprehension in maintenance tasks.

Teaching roles to novices has been found to improve learning outcome in procedural programming [3, 10]. In our future work we will investigate whetheher the same effect of improved programming skills applies to object-oriented programming, also.

6. ACKNOWLEDGMENTS

We would like to thank all those who volunteered to participate in the study. This work was supported by the Academy of Finland under grant number 206574.

7. REFERENCES

[1] J. R. Anderson. *Cognitive Psychology and Its Implications*. Worth Publishers, 5th edition, 2000.

[2] M. Ben-Ari and J. Sajaniemi. Roles of variables as seen by CS educators. In *The 9th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2004)*, pages 52–56. Association for Computing Machinery, 2004.

[3] P. Byckling and J. Sajaniemi. Using roles of variables in teaching: Effects on program construction. In P. Romero, J. Good, S. Bryant, and E. A. Chaparro, editors, *Proceedings of the 17th Annual Workshop of the Psychology of Programming Interest Group (PPIG 2005)*, pages 278–292, 2005.

[4] C. L. Corritore and S. Wiedenbeck. Mental represenations of expert procedural and object-oriented programmers in a software maintenance task. *International Journal of Human-Computer Studies*, 50:61–83, 1999.

[5] F. Détienne. *Software Design—Cognitive Aspects*. Springer-Verlag, 2002.

[6] Y. Kulikova. Roles of variables in functional programming. Unpublished Master's Thesis, Department of Computer Science, University of Joensuu, Finland, 2005.

[7] J. Peltomäki and P. Malmirae. *Java-ohjelmoinnin peruskirja*. Teknolit Oy, 1999.

[8] J. Sajaniemi. Roles of variables home page. http://www.cs.joensuu.fi/~saja/var_roles/. (Accessed Jun. 6th, 2005).

[9] 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.

[10] J. Sajaniemi and M. Kuittinen. An experiment on using roles of variables in teaching introductory programming. *Computer Science Education*, 15(1):59–82, 2005.

[11] J. Sajaniemi and R. Navarro Prieto. Roles of variables in experts' programming knowledge. In P. Romero, J. Good, S. Bryant, and E. A. Chaparro, editors, *Proceedings of the 17th Annual Workshop of the Psychology of Programming Interest Group (PPIG 2005)*, pages 145–159, 2005.

[12] A. Wikla. *Ohjelmoinnin perusteet Java-kielellä*. OtaDATA, 2003.