Report Series A

JOENSUUN YLIOPISTO

Teaching Programming: Going beyond “Objects First”

Jorma Sajaniemi and Chenglie Hu

Report A-2006-1

UNIVERSITY OF JOENSUU
Department
of
Computer Science

UNIVERSITY OF JOENSUU
DEPARTMENT OF COMPUTER SCIENCE
Report Series A

Teaching Programming: Going beyond "Objects First"

Jorma Sajaniemi and Chenglie Hu

Report A-2006-1

ACM K.3.2
ISSN 0789-7316
ISBN 952-458-801-3

Teaching Programming:
Going beyond “Objects First”

Jorma Sajaniemi
Department of Computer Science, University of Joensuu, Finland

Chenglie Hu
Department of Computer Science, Carroll College, WI, USA

March 3, 2006

Abstract

The prevailing paradigm in teaching elementary programming uses Java as the first programming language and the “objects first” approach as the conceptual basis. This approach has several shortcomings, e.g., high drop-out rates and poor skills in basic constructs like loops. This paper suggests an alternative approach that combines a strong start in basic constructs with early object-orientation. The alternative approach is also compared with the ACM Computing Curricula.

1 Introduction

The prevailing paradigm in teaching elementary programming uses Java as the first programming language and the “objects first” approach as the conceptual basis. The use of Java is motivated by its extensive use in industry and students’ wish to learn a “real” language that can “guarantee” them a job in future. The objects first approach is used to avoid negative transfer effects from procedural programming that make the transition to object-oriented programming hard. Thus the widespread use of object-oriented programming in software industry has lead to the abandonment of the previous teaching paradigm: Pascal as the first programming language and procedural programming as the conceptual basis.

However, most students do not become professional programmers but will need only basic understanding of programming and an elementary skill to use simple end-user programming tools. Moreover, the relatively high complexity of Java and the abstract nature of objects first approach have been criticized as demonstrated by recent panels on this topic in computer science education conferences [1, 2, 19].

This paper makes a literature survey on the problems of the current approaches, suggests an alternative approach, and compares it with the ACM Computing Curricula. The alternative approach combines a strong start in basic constructs with early object-orientation.

2 Current Problems

There are several shortcomings of the current “objects first with Java” paradigm. The objects first approach means that programming courses start with the introduction of objects that model some application domain, the attributes of these objects, the responsibilities of objects and their relationships with other objects, and finally the implementation of the responsibilities by the use of methods. As a result, learners have to work with abstractions for several weeks before they can base those abstractions to program code—another abstraction even though at a more concrete level. The high drop-out rates of objects first programming courses may be due to problems in acquiring correct understanding of the abstractions that those courses start with [14].

Modeling an application domain without knowledge of the techniques needed in the implementation of the model is like designing a bridge without knowing the limitations of steel constructs, or designing knitting models without knowing the limitations of knitting techniques. However, knitting is not taught by starting with knitting models but with practicing the use of knitting needles in order to first obtain a basic understanding of what can be done and how it is accomplished. Likewise, one can't expect engineers to grasp the meaning of actual constructs just by having them seated in front of a CAD screen; nor can one imagine that programming novices could design and comprehend models of application domains if they do not understand the constituents of the models. Only by knowing the basic building blocks and by understanding the basic techniques needed in combining them it is possible to make designs that can actually be implemented.

The question of the first programming language is not free of problems, either. Whereas Pascal was originally designed for educational purposes and was simple and consistent, Java is designed for professional use, uses cryptic notation that is not always consistent, and contains versatile class libraries that are too complex for novice use [2]. It is symptomatic that in a learning object designed to teach the concept of arrays [12] the tasks under the heading "test your understanding" require knowledge of Java syntax details rather than real understanding of the array concept. Similarly, in a Java-based CS1 course for academically diverse students [5] the purpose of many programming assignments is to introduce students to the functionality of components in a specific Java library rather than to promote understanding of object-orientation.

There is a considerable amount of evidence that novices learning programming have severe problems in understanding the basic concepts of programming (see [15] for a review). For example, the notion of a variable has been proven to be hard to understand; basic control structures like iteration are often misunderstood; and even the use of special notation like semicolons poses problems. A study in four universities [13] concluded that "many students do not know how to program at the conclusion of their introductory courses" and that "many students have not even acquired the technical skills needed for getting a program ready to run". It is no wonder that students have faulty mental models concerning objects, attributes, and methods [8, 9], when their mental models of much simpler structures like variables and basic control structures are often faulty.

A classic overview of programming pedagogy [20] notes that "One wonders, for example, about teaching sophisticated material to CS1 students when study after study has shown that they do not understand basic loops" and stresses the importance of teaching valid mental models because "if the instructor omits them, the students will make up their own models of dubious quality". The objects first approach tries to provide valid mental models of object-oriented design, class hierarchies etc but this is too sophisticated for novices who do not understand the basic building blocks of programs.

3 The Alternative Approach

The objects first approach tries to avoid the negative transfer effects by avoiding teaching procedural programming before object-orientation. At the same time it sacrifices concreteness of programs and uses abstractions that are overly hard to root on novices' existing knowledge [10]. There is, however, an alternative solution that starts with simpler—although yet hard for many learners—concepts: the concept of variables, their responsibilities (like keeping track of the number of input items), and the implementation of responsibilities by assignment statements and basic control structures (e.g., conditional, iteration). Only then the concept of objects and attributes, their responsibilities, and the implementation of responsibilities by methods are introduced. This "variables first, objects then" approach starts with more concrete concepts than the objects first approach but still avoids the introduction of procedural programming. Furthermore, the same basic ideas of responsibility and its implementation are applied twice: first to variables and then to objects.

The responsibilities of variables can be treated by the recently introduced notion of "roles of variables" [16]. The role set consists of a small number of roles like "stepper"—a role covering the notion of counting items. Roles give a vocabulary for responsibilities of variables and provide a sound basis for the variables first part. Roles belong to experts' tacit programming knowledge [18] and their use in teaching elementary procedural programming has been found to enhance learners' programming skills [3, 17]. In object-oriented programming, roles apply to attributes, local variables and method parameters. Thus the transition from variables to objects is conceptually simple and means only a shift in the abstraction level.

The set of basic control structures introduced in the variables first part of the course should include sequence, selection, and iteration but other control structures common to both object-oriented and procedural programming may be considered, also. For example, exception handling and concurrency (or multithreading) are central concepts in object-orientation [6] and their basic ideas can be introduced apart from objects and classes. Similarly, for the introduction of the common parts of methods and procedures we suggest a new notion, *named compounds* that covers parameter passing and recursion but that will not be used for structured design as such. Named compounds are motivated by the need for code re-use ("define once—use several times") and clarity ("separate different parts of the code") that are required both in object-oriented and in procedural programming. In object-oriented programming, named compounds turn into constructors and methods whereas in procedural programming they turn into hierarchical procedures and functions.

The variables first part of the suggested new approach may use some mini language in order to avoid complicated syntax requirements of, e.g., Java. When the basic programming constructs are mastered, a new language using a different notation may be introduced. This could even emphasize the unimportance of notational details. On the other hand, an educational situation might best be served by a language specially designed using pedagogic principles [7]. This supports the idea that a (new) educational programming language should be used throughout the approach.

The suggested approach provides students valid models of basic programming knowledge that can be applied both in object-oriented and procedural programming. It does not stress programming language features or any specific design technique to model the programming problem. Thus problems of any specific perspective to programming [4] can be avoided. The first course is intended to give a good understanding of programming constructs whereas the design and composition of larger programs—including notions such as encapsulation, inheritance, and polymorphism—is postponed until later courses in the curriculum.

4 Comparison with CC2001

ACM Computing Curricula CC2001 [11] offers several approaches to introductory courses: imperative-first, objects-first, functional-first, breadth-first, algorithms-first, and hardware-first. Each of these approaches consist of two courses (with an alternative three-course implementation in some approaches). In the same vein, our approach can be termed variables-first[1].

The course contents suggested in the previous section is not the same as the imperative-first approach of CC2001 which introduces the whole traditional procedural model; our suggestion avoids intentionally the procedural approach to program decomposition. In fact, our approach is closer to the algorithms-first approach where the basic concepts of computer science are introduced using pseudocode instead of an executable languages and which permits students to work with a range of data and control structures. However, in our new approach we assume that the programs are executable—even if written in some educational programming language that resembles pseudocode and has simple syntax.

Our "variables-first" approach does not try to give exact implementations of two or three introductory courses. However, a possible implementation—presented in the style of CC2001—consists of , e.g., two courses: Fundamentals of programming, and Abstraction mechanisms with the following syllabi.

Fundamentals of programming:

- Background: History of computing, overview of programming languages and the compilation process
- Simple data: Variables, types, and expressions; assignment
- Simple control structures: Iteration; conditionals
- Algorithms: Problem-solving strategies; implementation strategies; roles of variables
- Simple data structures: Arrays; records; strings
- Machine level representation of data: Bits, bytes, and words; binary representation of integers; representation of character data; representation of records and arrays
- Code re-use and functional decomposition: Named compounds; parameter passing
- Recursion: The concept of recursion; divide-and-conquer strategies
- Advanced control structures: Exceptions, concurrency
- Software engineering issues: Tools; processes; requirements; design and testing; risks and liabilities of computer-based systems
- Introduction to basic algorithmic analysis

[1]As a terminological detail, CC2001 should rather use the term "procedures-first" instead of "imperative-first" because the imperative-first approach covers not only imperative control structures (sequence, selection, and iteration) but also procedural abstraction. The term "imperative first" actually matches better our approach which starts with the common parts of object-oriented and procedural programming. For clarity, we have used a fresh name for our approach ("variables-first") although we would like to use the term "imperative-first" instead.

Abstraction mechanisms:

- Principles of encapsulation: Encapsulation and information-hiding; separation of behavior and implementation
- Abstraction in procedural programming: Procedures and functions; structured decomposition
- Abstraction in functional programming: Functions without variables; recursion over lists, recursive backtracking
- Abstraction in object-oriented programming: Classes and objects; methods; message passing; subclassing and inheritance; polymorphism
- Data abstraction: Classic data structures (list, stack, and queue); procedural implementation; object-oriented implementation
- Object-oriented design: Fundamental design concepts and principles; introduction to design patterns; object-oriented analysis and design
- Using APIs: Class libraries; event-driven programming; packages for graphics and GUI applications
- Software engineering: Building a medium sized system, in teams, with algorithmic efficiency in mind

CC2001 defines the core contents for computer science body of knowledge. The core consists of material that essentially everyone teaching computer science agrees is essential to anyone obtaining an undergraduate degree in this field. Core hours correspond to the in-class time required to present the material in a traditional lecture-oriented format. This time does not include the instructor's preparation time or the time students spend outside of class. Table 1 presents a comparison of the core hours of three CC2001 approaches (imperative-first, objects-first and algorithms-first) and the suggested "variables-first" approach.

5 Conclusion

ACM Computing Curricula 2001 states [11, Chapter 7]:

> Throughout the history of computer science education, the structure of the introductory computer science course has been the subject of intense debate. ... recommending a strategy for the introductory year of a computer science curriculum all too often takes on the character of a religious war that generates far more heat than light.
>
> In the interest of promoting peace among the warring factions, the CC2001 Task Force has chosen not to recommend any single approach. The truth is that no ideal strategy has yet been found, and that every approach has strengths and weaknesses. ... Moreover, we must encourage institutions and individual faculty members to continue experimentation in this area. Given a field that changes as rapidly as computer science, pedagogical innovation is necessary for continued success.

Table 1: Comparison of the amount of core hours in different approaches. Imperative-first (IF), objects-first (OF), and algorithms-first (AF) are suggested by CC2001; variables-first (VF) is our new suggestion. CC2001 total core hours in parentheses.

Topic	IF	OF	AF	VF
DS5 Graphs and trees (4)	2	-	-	-
PF1 Fundamental programming constructs (9)	9	9	9	9
PF2 Algorithms and problem-solving (6)	3	4	3	4
PF3 Fundamental data structures (14)	12	11	11	11
PF4 Recursion (5)	5	5	5	5
PF5 Event-driven programming (4)	-	2	3	3
AL1 Basic algorithmic analysis (4)	2	2	2	2
AL2 Algorithmic strategies (6)	-	2	4	2
AL3 Fundamental computing algorithms (12)	6	6	6	6
AL4. Distributed algorithms (3)	-	-	-	1
AL5 Basic computability (6)	1	1	1	1
PL1 Overview of programming languages (2)	2	2	2	2
PL2 Virtual machines (1)	1	1	1	1
PL3 Introduction to language translation (2)	-	-	2	-
PL4 Declarations and types (3)	3	3	3	3
PL5 Abstraction mechanisms (3)	3	3	3	3
PL6 Object-oriented programming (10)	10	12	8	11
AR2 Machine level representation of data (3)	1	-	-	1
AR3 Assembly level machine organization (9)	2	-	-	-
HC1 Foundations of HCI (6)	-	1	-	-
GV1 Fundamental techniques in graphics (2)	2	2	2	2
SP1 History of computing (1)	1	1	1	1
SP5 Risks and liabilities (2)	-	1	-	-
SE1 Software design (8)	4	4	4	4
SE2 Using APIs (5)	2	2	2	2
SE3 Software tools and environments (3)	2	2	2	2
SE5 Software requirements and specifications (4)	1	-	1	-
SE6 Software validation (3)	1	1	1	1
SE7 Software evolution (3)	-	-	1	-
Total core hours	75	77	77	77

In this paper, we have surveyed the problems of the current approaches to teaching elementary programming and suggested a new approach, "variables-first". This approach combines a strong start in basic constructs with early object-orientation. We have also sketched a two-course implementation of this approach and compared it with the Computing Curricula 2001 approaches. In future we plan to try this approach in a real classroom setting.

Acknowledgments

This work was supported by the Academy of Finland under grant number 206574.

References

[1] O. Astrachan, K. Bruce, E. Koffman, M. Kölling, and S. Reges. Resolved: Objects early has failed (Panel). In *Proceedings of the 36th SIGCSE Technical Symposium on CS Education*, pages 451–452, 2005.

[2] F. Bailie, M. Courtney, K. Murray, R. Schiaffino, and S. Tuohy. Objects first - does it work? (Panel). *Journal of Computing Sciences in Colleges*, 19(2):303–305, 2003.

[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 278–303. University of Sussex, U.K., 2005.

[4] H. B. Christensen. Implications of perspective in teaching objects first and object design. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education ITiCSE'05*, pages 94–98. ACM, 2005.

[5] J. Comer and R. Roggio. Teaching a Java-based CS1 course in an academically-diverse environment. In *Proceedings of the 33th SIGCSE Technical Symposium on CS Education*, volume 34(1) of *ACM SIGCSE Bulletin*, pages 142–146, 2002.

[6] F. Culwin. Object imperatives! In *Proceedings of the 30th SIGCSE Technical Symposium on CS Education*, volume 31(1) of *ACM SIGCSE Bulletin*, pages 31–36, 1999.

[7] B. Du Boulay, T. O'Shea, and J. Monk. The black box inside the glass box: Presenting computing concepts to novices. *International Journal of Human-Computer Studies*, 51:265–277, 1999.

[8] A. Eckerdal and M. Thuné. Novice Java programmers' conceptions of "object" and "class", and variation theory. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education ITiCSE'05*, pages 89–93. ACM, 2005.

[9] S. Holland, R. Griffiths, and M. Woodman. Avoiding object misconceptions. *SIGCSE Bulletin*, 29:131–134, 1997.

[10] C. Hu. Rethinking of teaching objects-first. *Education and Information Technologies*, 9:209–218, 2004.

[11] Joint Task Force on Computing Curricula. Computing curricula 2001. http://www.sigcse.org/cc2001, 2001. (Accessed Nov. 24th, 2005).

[12] London Metropolitan University. Arrays learning object. http://www.codewitz.org/demo/index.html, 2003. (Accessed Nov. 24th, 2005).

[13] M. McCracken, T. Wilusz, V. Almstrum, D. Diaz, M. Guzdial, D. Hagan, Y. Ben-David Kolikant, C. Laxer, L. Thomas, and I. Utting. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. In *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education ITiCSE'01*, pages 125–140. ACM, 2001.

[14] I. Milne and G. Rowe. Difficulties in learning and teaching programming—views of students and tutors. *Education and Information Technologies*, 7:55–66, 2002.

[15] A. Robins, J. Rountree, and N. Rountree. Learning and teaching programming: A review and discussion. *Computer Science Education*, 13:137–172, 2003.

[16] 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.

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

[18] 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. University of Sussex, U.K., 2005.

[19] G. R. S. Weir, T. Vilner, A. J. Mendes, and M. Nordström. Difficulties teaching Java in CS1 and how we aim to solve them (Panel). In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education ITiCSE'05*, pages 344–345. ACM, 2005.

[20] L. E. Winslow. Programming pedagogy — a psychological overview. *SIGCSE Bulletin*, 28:17–22, 1996.