Use of Morphisms as a tool to help learning object oriented concepts

Inés Friss de Kereki[1], José Crespo[2], and Javier Azpiazu[2]
1 Universidad ORT Uruguay, Facultad de Ingeniería,
Cuareim 1451, 11100 Montevideo, Uruguay
kereki_i@ort.edu.uy, WWW home page: http://www.ort.edu.uy
2 Universidad Politécnica de Madrid, Facultad de Informática
Campus de Montegancedo S/N, Boadilla del Monte, 28660 Madrid, España
{jcrespo, jazpiazu}@fi.upm.es, WWW home page: http://www.fi.upm.es

Abstract. Software design implies searching for and establishing an adequate morphism between the real world and the desired software. Morphisms establish correspondences between different domains while some properties are preserved, at the same time. It allows seeing different things as the same, taking the substitute image for the real one. The more adjusted to reality the morphism is, the better the system models the real situation. We propose the use of morphisms as a pedagogical tool in order to teach object-oriented concepts and also to promote better software design. We developed a course based on the explicit use of morphisms. Through experimentation, we compared the results with an equivalent course not using morphisms. From the results we may infer that using morphisms helps to develop strategies to analyze and to construct adequate software models.

1 Teaching and Learning Object Oriented Concepts

Education is no longer primarily the one-way transmission of information and knowledge [1]. Faculty must understand the different ways in which students learn, so they can adapt teaching styles to the learning style most effective for individual students, preparing students for a lifetime of learning [2]. Data or propositions, on one hand, and skills or procedures on the other, are taught in teaching systems. Those two learning ways are easily managed, weighed up and assessed [3]. However, as indicated by Pazos [4], students should be also prepared to synthesize, to set up and to contrast conjectures and to use their creativity.

Students should be taught how to think and act independently. This will allow them to gain more knowledge with increasing skill and dexterity [5]. Learning involves domain-specific and domain-transcending knowledge. The availability of prior knowledge is a crucial factor, but meta-cognitive knowledge such as task

knowledge, self-knowledge and strategic knowledge is also important to the learning process [6]. As Papert indicates, when students enter a new knowledge domain, they usually encounter a multitude of new ideas [3]. Pazos [4], quoting Papert, refers to the concept of "powerful ideas". Among them, it is important to emphasize morphisms, "ductions" (deductions, inductions, retroductions and abductions) and recursion. Morphisms establish correspondences between different domains while, at the same time, preserve and enforce some properties. Morphisms are more detailed and specific than the usual notion of abstraction. The use of morphisms allows seeing different things as the same, taking the substitute image for the real one.

In this work we propose the explicit use of morphisms as a tool to approach issues relative to the utility of powerful ideas, allowing an improvement in both quality and performance of the learning process. Particularly, we focused on the introductory programming courses, Computer Science 1 (CS1) in order to enhance the opportunities for the students to become successful in learning design and programming and, therefore, obtaining adequate models and programs.

Teaching object-oriented (OO) problem-solving and programming has proved to be more difficult than expected [7]. Many students find the conceptual issues involved in OO programming hard to understand [8]. One of the major problems when teaching an introductory course in OO problem solving and programming is the lack of suitable and proved methods to teach OO concepts and programming [7]. Bruce stresses that there is still insufficient data to evaluate how effective are pedagogical tools (like pedagogical IDEs, special libraries providing useful classes or microworlds) in introductory courses [9].

There are several approaches to teach programming courses. Some of them are basically related to the course organization: lectures versus lab work, individual versus collaborative work [10] or objects from the very beggining with supplemental instruction [7]. Other approaches, are oriented to methodology, for instance the use of extreme programming practices [11,12], CRC-Cards [13] or software tools, such as simulation [8] or visualization tools [14]. Besides, problem-solving capabilities, like structuring, abstraction and formalisation, planning and revising, are important [13]. McCracken *et al.* observed that students often skip the early stages in the problem-solving process and concentrate on implementation activities, rather than activities such as planning, design or testing [15].

As Huet *et al*. [10], we are actively involved in trying to enhance the students' learning experience through reflection on teaching approach and trying new ideas to aid students succeed. Use of morphisms as a tool might help to develop mental models and to give metacognitive support, as well as to promote planning activities and better software design.

2 Computer Science 1 and Morphisms

The course CS1 at Universidad ORT Uruguay introduces students to programming and to the paradigm of OO programming. Performance expectations are to identify, explain, and use classes and objects and to develop programs in an OO manner. The course applications are developed in Java. The summarised plan of CS1 is: week 1-3: variables, control structures and pseudocode; week 4: introduction to classes and

objects; standard classes; week 5-8: creation of classes, aliases, relationships between classes; week 9: inheritance; week 10-12: collections, exceptions, sort and search; and week 13-15: advanced use of collections. Each week of the course includes 4 theory hours (60 minutes each) and 2 practice hours (in laboratory).

The usual CS1 course includes lectures/demonstrations and separate laboratory sessions. It is based on learning theorical and practical contents taught with a fundamentally descriptive strategy. Our proposal uses morphisms to enhance the students' learning performance. Our hypothesis was that learning with morphisms improves the process of learning, by increasing students' skill to model and solve the assigned problems.

In a morphisms-based course, the concepts about morphisms are introduced in the 8th week. After that, programming examples and exercises are solved by focusing on morphisms concepts. In each example and exercise, the students analyze and propose a model. Then, each operation and representation is carefully studied. Using morphisms explicitly helps in elaborating specifications, as it requires determining which elements of the domain will match which elements of the models and which are the valid operations available. The gaps between the model and the real problem are detected and solutions are discussed. Practical examples include, for instance, modelling a temperature class, designing a system for house budget or representing withdrawals and deposits into a bank account.

In adittion, other examples are presented to reinforce the idea. On week 11 the 'JAM' [16] exercise is proposed with the intention that students discover the morphism themselves. This exercise is isomorphous to the well known game Tic-Tac-Toe. In the following weeks, additional work about morphisms is done. In each instance, an effort is made to establish relations with the original domain. For instance, the so-called "Year 2000 Date Problem" is analized, which provides an example of how a careful study of the behavior of operations in different domains should be a prerrequisite in modelling and programming. In this case, a basic calendar operation, "the next day operation" behaves undesirably in the digital domain because of an inadequate representation.

3 Experimentation and Results

This section aims to document an empirical comparison carried out with two different teaching methodologies, namely, standard and morphisms-based. We wanted to asses if the morphisms-based methodology gave students better skills in software design than the standard theory-practice courses.

Two student groups took part in the experiment. Students were randomly distributed and they belonged to the same age group (18 to 20 years old). They stated having no prior programming experience and not being currently employed. Group I (15 students) received the usual, standard course. Group II (16 students) received the same course plus theoretical material and exercises about morphisms. Solving strategies in Group II were based on morphisms. Each week, at least 20 minutes of a class were dedicated to these topics. Also, the strategy for solving problems was focused on detecting the morphism.

The use of morphisms hypothetically helps to develop a better model of a situation. The independent variable in our experiment is the morphisms training. The dependent variable –the one that indicates if the treatment had some effect- is the modelling capacity. This capacity is analysed in relation to the following three points: a) Model analysis: beginning with a given reality model, identify possible problems; b) Data representation for particular cases: analyse and define the representation of particular types of data; and c) Creation of a domains' model: representation of a domain, detecting principal classes and relationships as well as attributes and methods.

Two tests were given to each student in each group. Both tests had three questions each. The first test was given on week 8, the second at course's end (week 15). In each test, one question was aimed to each referred point (a, b and c) in the preceding paragraph.

Both tests were graded using ordinal scales. Each question was graded from 0 to 6. The samples were then compared using the Mann-Whitney and the Sign test [17]. In the first question, according to the Sign test, an improvement of reality grasping was detected in Group II. Regarding the second question, groups were found different (Mann Whitney; $\alpha = 0.05, 0.10$) in the first test. When test scores were analysed, Group I had a high proportion (80%) of high level results (4, 5 or 6 points), while Group II only had 7 students in these conditions (43.75%). In the second test, however, no significant statistical differences were found between the groups; it may be inferred that training helped to develop skills for adequate data representation in Group II. As to the third question, no differences were found in the first test, but in the second one Group II showed significant differences. 40% of Group I students got high values (4, 5 or 6), while 12 of the 16 Group II students (75%) got similar results.

Therefore, from the Sign test and Mann-Whitney [17] test results, we may infer that using morphisms allowed an improvement of skills in modeling a given situation, helped to represent data accurately and contributed positively to develop skills for constructing a domain model.

4 Conclusion and Future Work

The use of morphisms is presented in this paper as a useful tool to help developing learning strategies for analyzing and constructing software models. Through experimentation, it was found that students who participated in the morphisms-based course obtained better results in topics related to modelling than students of the standard course. A new experiment will be carried out in 2006 in order to try to confirm that these results can be replicated. Also, as an additional element, a software system for promoting model related skills based on the explicit use of morphisms is being developed and will be used in future courses.

It is proposed for future investigations to study, besides morphisms, the influence of ductions on learning. In this way, some conclusions might be drawn as to which of these two powerful ideas has a larger positive impact on learning, or whether both together interact, for instance in an additive or multiplicative way.

5 References

1. T. Bentley, Learning beyond the classroom, *Educational Management & Administration*, **28**(3), 353-364 (2000).
2. National Academy of Engineering, *The Engineer of 2020. Visions of engineering in the new century* (The National Academies Press, Washington DC, 2004).
3. S. Papert, *Mindstorms* (Basic Books, USA, 1980).
4. J. Pazos Sierra, Enseñanza del futuro: a grandes males pequeños remedios. Technical Report, Universidad Politécnica de Madrid, España (unpublished), 2001.
5. P. Jackson, *Práctica de la Enseñanza* (Ed. Amorrortu, Buenos Aires, 2002).
6. F. Dochy, C. de Rijdt, and W. Dyck, Cognitive prerequisites and learning, *Active Learning in higher education*, **3** (3) 265-284 (2002).
7. L Kalling and M. Nordström, Teaching OO Concepts - A New Approach, in *Proc. of the 34th. ASEE/IEEE Frontiers in Education Conference FIE 2004* (Savannah, 2004), pp. F3C6-F3C11.
8. M. Esteves and A. Mendes, A simulation tool to help learning of object oriented programming basics, *Proc. of the 34th ASEE/IEEE Frontiers in Education Conference* (Savannah, 2004) pp. F4C7-F4C12.
9. K. Bruce, Controversy on how to teach CS1: a discussion on the SIGCSE-members Mailing List, *Inroads- The SIGCSE Bulletin* **36**(4), 29-35 (2004).
10. I. Huet, O. Rocha, J. Tavares and G. Weir, New challenges in teaching introductory programming courses: a case study, *Proc. of the 34th. ASEE/IEEE Frontiers in Education Conference FIE 2004* (Savannah, 2004), pp.T2H5-T2H9.
11. V. Jovanovic, T. Murphy and A. Greca, Use of extreme programming (XP) in teaching introductory programming, *Proc.of the 32th. ASEE/IEEE Frontiers in Education Conference FIE 2002* (Boston, 2002), p. F1G-23.
12. T. Ahern, Work in Progress: Effect of instructional design and pair programming on student performance in an introductory programming course, *Proc. of the 35th ASEE/IEEE Frontiers in Education Conference FIE 2005* (Indianapolis, 2005), pp. F3E11- F3E12.
13. J. Börstler and C. Schulte, Teaching object oriented modelling with CRC-Cards and roleplaying games, *Proc. of the 8th IFIP World Conference on Computers in Education WCCE 2005* (Cape Town, 2005).
14. A. Virtanen, E. Lahtinen and H. Järvinen, VIP a Visual Interpreter for Learning Introductory Programming with C++, 5th Koli Calling conference (Finland, 2005).
15. M. McCracken, V. Almstrum, D. Diaz, M. Guzdial, D. Hagan, Y. Ben-David, C. Laxer, L. Thomas, I. Utting and T. Wilusz, A multi-national, multi-institutional study of assessment of programming skills of first-year CS students, *ACM SIGCSE Bulletin*, **33** (4) 125-140 (2001).
16. Miscelánea de cuestiones matemáticas relacionadas con el taller. http://www.juntadeandalucia.es/averroes/iesarroyo/matematicas/taller/aspectosweb/aspectosweb.htm. (December 26, 2005).
17. R. Mason and D. Lind, *Statistical Techniques in Business and Economics*- 8th edition (Richard Irwin Inc, USA, 1993).