Computer Science Education
Vol. 15, No. 2, June 2005, pp. 123–142

PBL and Computer Programming — The Seven Steps Method with Adaptations

Esko Nuutila*, Seppo Törmä and Lauri Malmi
Helsinki University of Technology, Finland

Problem-Based Learning (PBL) method emphasizes students' own activity in learning about problems, setting up their own learning goals and actively searching for and analyzing information. In this paper, we describe and discuss our experiences on applying PBL, especially the seven steps method widely used in medical faculties, in an introductory computer programming course. We explain how the method is implemented, give examples and identify different kinds of PBL cases, and describe how the method is supplemented by other learning methods in our course. According to our experience, the PBL method increases the commitment of the students which results in a significantly lower drop-out rate: the average is 17% versus 45% in our traditional programming courses. In addition to computer programming, students also learn generic skills related to group work, collaborative design work, independent studying, and externalization of their knowledge.

1. INTRODUCTION

In 1966 McMaster University Medical School began to pioneer a new approach to medical education called Problem-Based Learning (PBL). This approach has subsequently spread to other medical schools and educational institutions worldwide. As the basic approach is not specific to medical education, it has been gradually applied in many other fields ranging from architecture and mechanical engineering to social work and law.

In PBL the learning of voluminous and fragmented material is centered around the types of problems that professional practitioners in the field would encounter in their day-to-day work. Thus, a student faces the material in a more integrative and motivating manner. This approach is not new; it is closer to the way learning took place before the advent of classrooms and organized education. Even in classrooms, good teachers everywhere have used elements of problem-based learning to motivate and activate the students, and to improve the transfer of the knowledge to real-world situations.

*Corresponding author. Esko Nuutila, Helsinki University of Technology, Laboratory of Information Processing Science, P.O. Box 5400, FIN-02015 HUT, Finland. E-mail: enu@cs.hut.fi

ISSN 0899-3408 (print)/ISSN 1744-5175 (online)/05/020123-20 © 2005 Taylor & Francis Group Ltd
DOI: 10.1080/08993400500150788

The effectiveness of PBL has been studied over the years. Norman and Schmidt (1992) review the evidence gathered. They conclude that while there has been no evidence that PBL results in an improvement in the results of standard examinations (or may even slightly lower them), "it may foster, over periods up to several years, increased retention of knowledge", "may enhance both transfer of concepts to new problems and integration of basic scientific concepts into clinical problems", "enhances intrinsic interest in the subject matter", and "appears to enhance self-directed learning skills."

Various specific ways for implementing PBL exist. Schmidt (1983) presents the *seven steps method* of PBL used at Maastricht University. The specific aims of this method is to foster learning by (1) *activating the prior knowledge* of students about the topics to learn, (2) connecting the learning to *specific* problem situations that might occur in practice, and (3) making the students to *elaborate* the material that they have learned.

In this paper we describe our experiences since 1999 of using the seven steps method of PBL in an introductory programming course in Helsinki University of Technology. Each year there has been about 30 first-year university students coming from the study program of information networks, a multi-disciplinary program combining information technology, social sciences and business management. About half of these students are female.

The goal of the course is to learn the Java programming language to the extent that the students can independently implement non-trivial Java applications (e.g., ones with a graphical user interface). In our implementation of the seven steps method, students meet once-a-week in groups to discuss real-world cases. The role of a case is to present something that *requires explanation or solution*, and that will lead the group to define learning goals. A new case is opened every week and closed next week after each member of a group has *independently studied* to achieve the learning goals. In one semester students process ten cases, each focusing on some important aspect of programming. The PBL cases and sessions replace the traditional lectures.

In our course, PBL is supplemented with other learning methods. Practical *programming assignments* and a personal *programming project* form an important part of the course. In addition, the students write *essays*, draw *concept maps* (Novak & Gowin, 1984), and collect all material they have produced in a *portfolio*.

In a questionnaire made in autumn 2002 the students reported that group work in PBL improved the motivation, provided emotional support, and gave a social context for the course. A measurable benefit of this learning method has been a drop-out rate much lower than in our traditional programming courses, 17% versus 45%. Both the students of the PBL course and the students of the traditional programming courses take subsequently the same advanced course in Java programming. The average scores for the students coming from the PBL course have been slightly better, but the difference is not significant.

From the teacher's point of view, the close interaction with the students in the PBL sessions enables rapid identification of difficulties in learning and spotting of problems in the course arrangements. Further, the PBL method seems to promote learning of new concepts and mechanisms as well as problem solving and program design.

PBL is a term that is used for a wide range of practical teaching approaches, and the role of the problems varies in these approaches. Firstly, according to (Schmidt, 1983) PBL refers to learning that is stimulated by descriptions of real-world problems. Students do not necessarily try to solve a problem but rather define learning goals to better understand it. Our PBL course is primarily based on this approach. Secondly, PBL is often used to mean a learning method in which students solve practical problems and the learning happens as a side effect of the solution process. This has also been called problem-oriented learning. Thirdly, when the size of the problems becomes larger, the approach becomes project-based learning.

Other applications of PBL in computer science courses have been reported in the literature. In the University of Sydney, foundation CS courses have been implemented with an approach that consists of two project like cases per semester (Fekete, 1998; Greening, 1997; Kay et al., 2000). The authors define PBL as "learning by solving a large, real-world problem." In Linkoping University PBL has been used in a course that integrates computer science studies with other engineering disciplines (Lambrix, 1998). This course is based on a single large scenario that is processed through four major themes. The role of problems in these courses differs from that in our course; in our classification these approaches fall more in the category of problem-oriented or project-based learning. However, similar benefits are reported: learning generic skills, familiarity with group work, and ability to deal with vaguely specified problems.

In Section 2 we present the PBL method and how we have implemented it on our course. We give examples of PBL cases that we have used. In Section 3 we list the observations we have made during different versions of the PBL course between 1999 and 2003. In Section 4 we present the other learning methods that were used in addition to the PBL method. Finally, we discuss the pros and cons of the method in Section 5 and identify such elements of programming skill that are well supported with the PBL method, as well as such elements which require different learning methods. The Appendix contains the full list of cases and the programming assignments we have used in our course.

2. IMPLEMENTATION

In this section, we describe the use of PBL in our programming course. We start by giving a short overview of the course and the variety of learning methods used in addition to PBL. Then we describe the PBL method in detail and give examples of PBL cases used.

2.1. Course Overview

The course is meant for first year university students, who have no previous programming experience. The goal of the course is that the students learn basic object-oriented programming in the Java language and are able to independently construct Java applications with graphical user interfaces.

Traditionally, a series of lectures has created the structure and rhythm for learning in our programming courses. The conceptual and factual content of a programming language, and elements of the programming skill are presented in the lectures. In this course, the PBL cases have a similar role. The course includes ten different PBL cases, approximately one case per week.

Since programming is a skill, it needs to be rehearsed. Thus, already in the first course version (autumn 1999) we supplemented the PBL cases with programming assignments and a personal programming project. Solving the programming assignments is supported by weekly meetings with the course assistants.

In addition, students write essays and draw concept maps about key mechanisms of the programming language. Based on the material they have produced in these tasks, they finally prepare a portfolio, in which they summarize and reflect their learning. The students are evaluated based on the programming assignments, the programming project, an exam, and the portfolio.

As the reader can guess, we do not believe PBL to be a magic solution for learning programming. However, PBL has a more central role than other learning methods mentioned above. It is the main organizing method in the course, and the weekly PBL sessions establish a social organization among the students and the teachers. We explain the motivation and the role of the other learning methods below.

2.2. PBL: The Seven Steps Method

Several concrete implementations of PBL exist. We have adapted our method from the seven steps method described in (Schmidt, 1983).

The students are divided into groups of 7 – 10 persons. Each PBL group is supervised by a tutor, whose primary role is to act as a domain expert, and the secondary role is to be a facilitator of the group process. The tutors are either faculty members or students who have taken the same PBL course earlier.

The tutor is mainly passive during the PBL session but is ready to answer any questions. However, if the discussion of the group takes a wrong track or if clearly false conceptions start to dominate, the tutor should correct the situation, for instance, with appropriate questions. If the activity in the group is unbalanced or unproductive, the tutor tries to improve the behavior of the group. Our experience suggests that if the tutor becomes too active, the PBL session starts to resemble a lecture, and the students begin to see the tutor as responsible for their learning.

Each PBL group meets once a week in a three hours PBL session. A group needs a meeting room equipped with a white board or a flip chart. Self-stick notes and white board markers are also provided for all students.

The meeting starts with the closing session of the previous case, if any. Then a new case is opened. The processing of the case goes through the sequence of steps shown in Figure 1.

Steps 1 and 2, examination of the case and identification of the problem, are straightforward. Only in the first few cases the students may have difficulties in finding a title for the case. The tutor may tell that the wittiness of the title is not important.

In step 3, brainstorming, the goal is to connect the case with the previous knowledge and experience of the students. It is essential that the students do not criticize each others — or their own — ideas. In fruitful brainstorming there are parallel and freely combining streams of ideas. In our experience, brainstorming is usually the least problematic step of the method.

Step 4, sketching of an explanatory model, has been less satisfying. The students usually confine themselves to regrouping the self-stick notes on the white board. Seldom any significant structural or conceptual reorganization of the material takes

Opening session – half an hour, in the group
Step 1: Examination of the case. The group gets familiar with the case material. Step 2: Identification of the problem. An initial title for the case is specified. Step 3: Brainstorming. The students present their associations and ideas about the problem to find out what is already known and how does the case relate to the previous knowledge. The ideas are said aloud and written on self-stick notes, which are organized on a white board. Step 4: Sketching of an explanatory model. An initial version of the explanation for the problem is constructed and most important concepts and their relations are identified. Step 5: Establishing the learning goals. Those parts of the explanatory model that are mysterious, fuzzy, or simply unknown are identified and the central ones are chosen as learning goals for the group.
Study period -- one week, each student working independently
Step 6: Independent studying. Each student independently studies to accomplish *all* learning goals. This phase includes information gathering and usually a substantial amount of reading (e.g., 50--150 pages).
Closing session -- one to two hours, in the group
Step 7: Discussion about learned material. Equipped with the newly acquired knowledge, the group reconvenes to discuss the case. The discussion includes *explanation* of central concepts and mechanisms, *analysis* of the material, and *evaluation* of its validity and importance.

Figure 1. The sequence of steps used to process each case

place. Recently, we have experimented with concept maps in sketching the model. The results have been encouraging: new abstractions and relationships emerge among the ideas and less fruitful ideas are left out. It is worth studying whether the tutor could take a more active role in guiding the sketching process. It is important, however, that the group, and not the tutor, provides the content of the explanatory model, since it should reveal what weaknesses or gaps there are in the current understanding.

In step 5, establishing the learning goals, the tutor should encourage the students to make the learning goals as precise and concrete as possible. Generally, the students are able to identify relatively good learning goals but often look for the tutor's approval for the goals. Finally, the group should identify material for independent studying (books, articles, and web sites), a task where the tutor can help.

Step 6, independent studying period, is the most crucial of the steps. If the students fail to study enough, the PBL method does not work. The importance of self study should be emphasized to the students from the beginning and the morale should be kept high throughout the course. In addition, this should be taken into account when designing the syllabus. There should be enough time reserved for independent studying.

The quality of step 7, discussion about learned material, is directly connected to the amount of self study. Optimally, each student is so familiar with the facts, concepts, and mechanisms related to the learning goals that the discussion is mostly focused on reflection on the meaning, importance, and use of learned material. In the other extreme, if the step 6 has failed, the group may even seem to know less about the subject than in the opening session.

The material should not be fixed too rigidly in step 5. It is beneficial if students use different material during independent studying. When multiple viewpoints exist, the quality of the discussion in step 7 will greatly improve. The discussion will be better motivated and more thorough. The students' meta-cognitive skills — evaluation of difference sources, resolution of conflicting viewpoints, and so on — will be rehearsed. The whole method will be more robust since errors or weaknesses in source material are more likely to be revealed and corrected.

2.3. Different Kinds of Cases

The cases differ not only in their content, but also in the form. We have identified at least three forms of cases:

1. Knowledge-oriented cases. The "classical" PBL cases belong to this category. The students are presented some curious real-world event or problem that creates a cognitive dissonance and requires explanation. If this happens, the students will have an intrinsic motivation to understand the causes and mechanisms behind the case.
   This kind of cases can be used to make the students learn important concepts, constructs, or factual knowledge about programming. An example is to learn the concepts of class and object in the Java programming language.

2. Design cases. These are adapted PBL cases that work well in programming courses and would probably work as well in any engineering-oriented domain. The PBL group is presented a complicated real-life design problem, typically too challenging for them to solve completely. In addition to training the problem solving skills and demonstrating the need to define new concepts and abstractions, the case can also lead to identification of gaps in the knowledge. Our PBL groups often do surprisingly well in this kind of cases.
   The learning goals identified in design cases usually differ from those of knowledge-oriented cases. Sometimes the design itself suggests some areas where learning is needed but typically the learning goals deal with the completion of the design or implementation of some part of it.
3. Analytical or diagnostic cases. This kind of cases were motivated by the constructivist learning methods where new learning begins from the identification and confrontation of students' previous misconceptions about the topic to learn (Thagard, 1990). (This has been a successful method to teach elementary physics where a conceptual change is often needed and the naive model of, for example, motion must be confronted to make room for new and more powerful model).
   An example of this kind of case is to present the students a seemingly correct but subtly erroneous computer program and the output of its execution. It is essential to choose a program that a student with some common misunderstanding would consider correct and whose output clearly shows it to be incorrect. The idea is that when the students try to understand the situation, the misunderstanding is revealed and as a result, the students will revise their mental models. These kind of cases, however, have not turned out to be successful. If there is even one member in the PBL group that does not have the misconception, she or he will quickly explain the situation to the others. Moreover, even if all students have similar misconceptions, some may rapidly identify them, while the others may completely miss the point. In our experience, individual diagnostic tasks should be used instead of group level diagnostic cases.

Currently our course thus consists of a mix of knowledge-oriented cases and design cases. Below are some examples of both. An example syllabus listing all cases is in the Appendix.

2.4. Examples of Cases

2.4.1. Case: Plato's Theory of Eternal Forms vs. Computer Games. This case is presented in the beginning of the course, when the students have no knowledge about objects and classes, which are the fundamental concepts in object-oriented programming. However, all students have taken an elementary course on philosophy in the high school and probably heard about Plato's theory of forms.

The case description contains a short summary of Plato's theory of forms as follows:

According to Plato, in addition to the perceptible world there exists another world, the world of forms. For instance, when we draw triangles in the sand, they all differ from each other. They have properties that are not essential to triangles, like the thickness of the lines. We can think that the drawings are not real triangles, but more or less accurate reflections of the form or idea of a triangle. This form is immaterial and eternal. For everything in the perceptible world there is a corresponding form: the form of a table, the form of a man, the form of beauty. According to Plato, the world of forms is more perfect and real than the perceptible world.

In addition, the case description presents a popular computer adventure game. The game contains creatures, friendly or hostile. One creature represents the player. Each creature has several properties like strength and wisdom. In addition, the game world contains items like weapons, food, jewels, and keys that the creatures can manipulate.

The tutor usually tells the students that the Platonic forms correspond to classes and the concrete things to objects in the Java language. The main objective in the case is to discover the distinction between a class and an object belonging to that class. In brainstorming, the students usually identify many possible classes and objects in the game, but sometimes confuse them with each other. The typical learning goals are related to classes and objects in the Java language.

In the closing session next week, the students have better understanding of the difference between an object and a class, and they may even be able to model parts of the game world in Java.

2.4.2. Design Case: A Computer File System. This case is presented a couple of weeks later, when the students already know some essential programming concepts. The goal is to introduce hierarchical object structures and recursion and to make a distinction to the class hierarchy. The case description is the following:

A computer file system consists of files that are of two different types: 1) documents and 2) folders. A folder can contain a set of files (either documents or folders) and each file belongs to some folder. Typical operations to files are move, rename, copy, and delete. The size of storage required by a file can be presented to the user. How can we model such a file system by Java-classes? What variables and methods are required?

The problem here is to find a minimal set of classes — only three are needed — and to design the file operations. The concept of recursion, which is not familiar to the students, is needed when copying folders and counting the size required by a folder. In the opening session, the students realize the need for recursion, but cannot yet formulate it. Further, they seem to have difficulties in understanding the idea of objects that refer to other objects and form a tree-like structure. By the closing session, the students have gained a rather fluent understanding of object hierarchies and recursion.

3. OBSERVATIONS

The PBL course has been given five times starting in 1999. Each year there has been about 30 first-year university students coming from the study program of information networks. About half of these students are female. The main observations are described below. For the years 1999–2003 we have collected drop-out rates. After the autumn 2002 we gathered information from the students about the use of time to different learning methods and subjective feelings about the usefulness of different methods.

3.1. Drop-out Rate and Quality of Learning

In years 1999–2003 the average drop-out rate was 17% ranging from 0% to 29%. The highest drop-out rate, 29%, was in the first course year 1999, when some of the course arrangements failed. After the course arrangements stabilized, the average drop-out rate has been 14% ranging from 0% to 22%. This is in a sharp contrast with our traditional programming courses where the average drop-out rate during the same period was 45% ranging from 41% to 51%.

In computing the drop-out rate, we included only those students that submitted at least one assignment in the course. In the traditional courses the enrolment includes many students who do nothing in the course.

In our subjective judgment, the quality of learning has been high in the PBL course. The students that pass the course generally submit good projects and are able to describe the implementation in a manner that shows good conceptual understanding. However, since the PBL course differs from the traditional courses in the structure, the student profile, and the evaluation method used, we cannot present any statistically valid data about the overall quality of learning compared to the traditional courses.

Both the students of the PBL course and the students of the traditional programming courses take subsequently the same advanced course in Java programming. In spring 2003, the average scores in this course where 2.92 (in a scale from 0 to 5) for the students coming from the PBL course and 2.78 for the others. In spring 2004, the corresponding scores where 2.83 and 2.44. Thus, the average scores for the students coming from the PBL course have been slightly better.

3.2. Group Work

Being a member of a group is an essential aspect of the PBL method. It seems to have a clear positive impact on learning. In a questionnaire made in autumn 2002 the students reported that group work in PBL improved the motivation, provided emotional support, and gave a social context for the course.

Being a member of a group of students of equal initial skills and equal difficulties helped the members to cope with the anxiety created by the challenging goals and contents of the course. In the traditional course, many students face the difficulties alone and are more likely to give up. Students working in a group will see that others have similar problems, a realization that in itself can be relieving. The members of a group can also help each other to understand difficult issues and provide emotional support to each other.

In addition, passing through a difficult course together also seems to create a strong bond among the students and they continue to work together and to support each other also in their subsequent studies.

It should be noted that in our PBL method the group work is only a tool that aids the members of the group in their individual learning. All external results in the course are produced individually: the examination, the portfolio, the programming assignments, and the programming project. Therefore, the problem of group-work called "free riders", i.e., members that avoid responsibility or contribution, does not arise in a same way as in the groups that produce common results. Naturally, the activity of different students in the participation to discussion varies, but that has not been a major problem in our course.

3.3. Individual Work

Many students use too little time for the independent studying for the closing session (step 6). To create and maintain strong morale in the PBL group throughout the course is the most challenging aspect when arranging a PBL course. The tutor should give the group clear feedback about the level of learning they have achieved. If the amount of individual studying is not sufficient, it should be indicated to the students. Feedback should be given also after the closing session: students often tell that they would like to know if they have studied enough. It can be more difficult to know this in PBL courses than in traditional courses, since learning goals are not specified as some particular amount of material to read.

3.4. Finding Good Cases

It has been somewhat more difficult — although not overwhelming — to find good cases than we thought in the beginning of the first PBL course. This difficulty is caused by two facts. First, the cases should be presented in concepts familiar to the students. However, the students have no previous knowledge about programming; the programming concepts are not familiar even from other contexts.

Second, the professions involving programming work do not provide a ready source of cases, as do some other professions. In the field of medicine, for example, the recorded descriptions of patients' visits to a doctor often provide a rich source of PBL cases. Programming is an engineering profession and therefore mostly project-oriented; it does not automatically decompose into small cases that could be summarized in a few lines of text.

3.5. Teachers' Point of View

As a result of the close interaction with students, the teachers get to know the students better. The interaction and close monitoring of the progress of the students improves the motivation of the teachers. Moreover, the feedback about the contents or the arrangements of the course comes quicker and is more relevant. The problems are identified earlier and it is easier to respond to them quickly. In addition to improving the on-going course, this also creates a better ground for the future development of the course.

4. ADAPTATIONS

In this section, we present the adaptations that we have made to the plain PBL method. Complementary learning methods were required to address the need to develop programming skills and to improve conceptual understanding. Over the years, the order in which the material has been presented to the students and the staffing of the course have also changed, as described below.

4.1. Programming Assignments and Programming Project

In the beginning our initial idea was simply to teach programming using the method of problem based learning. However, soon after we began the planning of the course syllabus for the first time, we realized that since programming is a skill, it needs to be rehearsed. We considered having group exercises, which might have been more in the spirit of PBL. However, there is a danger that some members of a group will end up doing most of the programming and the others would not learn the necessary practical skills well enough.

In the end, it was decided to have a set of individual programming assignments during the course, and an individual programming project towards the end of the course. This arrangement has always been the most natural and least problematic addition to the PBL method.

However, recently we have wondered if the work needed to complete the programming assignments might interfere with the amount of studying for the PBL closing sessions. The student questionnaire after the autumn 2002 course indicated that students used on average 5 hours/week on programming assignments whereas they used only 2.1 hours/week for other studying for the PBL closing sessions.

4.2. Essays, Concept Maps, and Case Reports

In the first version of the PBL course it was observed that students had difficulties in learning some abstract concepts such as, “object” or “class”. We gave them an additional assignment to write an essay about these concepts. Writing an essay turned out to clarify the students’ conceptual understanding considerably.

During the subsequent course versions we have paid more attention on choosing the PBL cases and the order in which the difficult concepts have been presented. Thus, the students have been able to learn the abstract concepts without major difficulties in the PBL cases. Consequently, the original motivation for the use of essays has largely disappeared.

However, we have observed that it is difficult to create meaningful PBL cases of some technical concepts of Java programming language such as exceptions, threads, or event handling. Characteristic to these concepts is that they deal with non-obvious design choices made by the creators of the language. Essays have turned out to be successful method of learning these concepts.

One problem with essays is that some students try to hide the lack of conceptual understanding behind verbose rhetoric. To make the students focus on the concepts and their relations, in autumn 2001 we encouraged the students to construct concept maps (see, for example, Novak & Gowin, 1984) as an alternative to essays.

The experiences on concept maps have been solely positive. The quality of the maps has been better than we expected. It is much easier to see the misunderstandings and gaps in the knowledge in a concept map than in an essay. An additional advantage is that copying concept maps from the Internet is more difficult than copying material to essays.

In autumn 2000, we experimented with PBL case reports. For each PBL case, each student wrote a short description of the opening session, the individual learning period, and the closing session. The goals were to make the students reflect their learning and to make them work in a more disciplined manner. Apart from a couple of exceptions, the case reports were superficial and contained no real reflection. The costs of writing and reading the case reports were considered much higher than the benefits that were gained. We therefore decided to drop them from the subsequent courses.

4.3. Demonstrations and Supervised Laboratory Times

In the first two course versions, the students had difficulties in practical skills such as using editors and other programming tools. Their programming habits were ineffective, e.g., writing the whole program before the first attempt to compile it.

Starting from autumn 2001, we have been arranging programming demonstrations to give examples of using the programming tools efficiently. These demonstrations have stimulated student questions more than traditional lectures. They have often turned into spontaneous and effective question and answer sessions.

Due to the problems that many novice students have with independent programming work, we introduced supervised laboratory times in the autumn 2001. This addition has been successful. There has been a real demand for this kind of assistance; students have been more than eager to ask the course assistants for advice.

4.4. PBL and the Presentation of Object-oriented Programming

The order in which different aspects of the domain of study are introduced is important in the PBL method. In a successful PBL session the students can connect the case to something that they already know. If they can not, the result is a silent and confused PBL session that is damaging to the motivation of the students. In an introductory programming course this problem is emphasized due to the abstract nature of the domain and the general lack among the students of any knowledge about internals of software systems. In our PBL course we encountered this problem specifically with the order in which different aspects of an object-oriented programming language were introduced.

In the first course version 1999, the learning of Java started from algorithms and control structures and only later on proceeded to object-oriented programming.

The first case described the Russian peasant's method of multiplication (a method that reduces multiplication of large numbers to doubling, halving and addition operations) with the aim of introducing the concept of algorithm. Unfortunately, the students failed to see the point and did not manage to specify any meaningful learning goals without the assistance of a tutor.

Later, when the students had gained some understanding of program control with simple scalar and array variables, the concepts of object-oriented programming were introduced in another case. That was not a great success either. After the students had finally got familiar with the details of control, they appeared to find it difficult to grasp more abstract entities such as objects and classes.

We had a hypothesis that concepts dealing with algorithms and program control are remote to the previous knowledge of the students and that objects and classes would be easier for students to understand. The general trend in the new Java text books conforms with this hypothesis: they increasingly use so-called objects-first approach as described in the ACM Computer Curricula 2001 (ACMCC, 2001). This approach has another benefit too. Since the basic knowledge about objects and classes would enable the use of design cases, the object-first approach could make it possible to use design cases earlier on and thus offer more freedom in the design of the syllabus.

In autumn 2000 we switched to objects-first approach: the first cases dealt with the basic concepts of object oriented programming, then focused on the methods of objects, and finally on the algorithms executed in the methods. After this change, the quality of learning generally improved and the confusion felt by the students reduced. We did not encounter really unsuccessful PBL sessions anymore.

It is, however, not clear that these improvements are due to the switch to objects-first approach. At the same time many old cases were dropped or modified, and new cases were introduced. In fact, one of the most successful cases ever is the "robot in a maze" (see Case 4, in Appendix). It essentially deals with the design of an algorithm that a robot could use to find a way out of a maze. The students find this case inspiring and end up with relatively good algorithms. This case works already in the beginning of a course, since it does not require knowledge of objects and classes.

We have come into conclusion that our hypothesis about the superiority of objects-first approach is too simplistic. It is possible that the real flaw of the Russian peasant's case mentioned above is different: Before the students had designed any algorithms ever, they were presented a ready-made algorithm and were then expected to recognize abstract algorithmic concepts such as loops, conditionals, and termination. This is a difficult task for someone with no previous experience in the design of algorithms. However, the students seem to perform quite well in tasks of designing specific algorithms, and after a couple of such tasks the abstract algorithmic concepts are much easier to recognize.

As a summary, it is not obvious that objects-first approach is necessary, or even the best one, when teaching Java using the PBL method. It is clear, however, that an important question to consider when designing the syllabus of a PBL course is how well the students will be able to connect each of the cases to their previous knowledge at the time that case is presented.

4.5. Students as Tutors

In the first two years of the course all the tutors were members of the faculty. In the autumn 2001 we began to use the best students from the previous years as tutors. So far the experiences have been good. This is not a unique finding as similar experiences are reported in (Ventura, 2003).

When compared to the alternative of using faculty members or graduate students as tutors, previous years students have some distinct advantages. They are often eager to become tutors. There are plenty of them and they are not as expensive as the members of the faculty. They have gained first-hand experience of the PBL method and are thoroughly familiar with the PBL process. While their technical knowledge about the area is not as deep as that of the faculty, in practice it has proven to be quite sufficient, and sometimes even more pertinent to the needs of the students. Moreover, they are in a position to better understand the conceptual or technical difficulties that novices face in the course. They are also familiar with different kinds of practical problems, for example, those related to the use of the student computer systems. Finally, in our experience, they generally can handle the problems in the group process quite as well as the members of the faculty.

4.6. Portfolio and Evaluation

We do not evaluate the PBL sessions, since that would probably have a negative effect on the spirit of the group work. The evaluation has been based on the deliverables that students have produced in the other learning methods as well as on a small exam. Since the autumn 2002, the evaluation of the students was based on the programming assignments (40%), the programming project (30%), a small exam (10%), and a portfolio (20%).

In the portfolio, the students collect their answers to the programming assignments, their programming projects as well as all learning material that they have gathered for the PBL cases. At the end of the course, the students write a summary of the portfolio (in HTML) for evaluation.

5. DISCUSSION

We define the concept “computer programming” as a skill of designing, implementing and analyzing computer programs of various scales. In this section we discuss the difficulties that novice students face when learning computer programming and how the PBL method relates to these difficulties.

First, a novice needs to acquire a large number of new, abstract concepts and mechanisms in order to have even a rudimentary programming skills. Examples are "procedure call", "module", "declaration", "parameter", "exception", "interface", "object", and "class". These concepts are interdependent and often describe different aspects of the same dynamic or structural mechanisms. These mechanisms relate to the programming language (for example to parameter passing) and to the data structures and algorithms (for example, to the structure of and operations on a binary tree).

Second, programming requires continual use of problem solving and design skills. Novice programmers seldom have any experience of this kind of cognitive activity. In other academic areas, e.g., mathematics and physics, the student tries to solve small problems defined by the teacher. There is typically only one correct solution, and the student does not use the solution anywhere. In programming, the problem is usually much larger and has a large number of possible solutions, some better and some worse. To design a program, the student must self define and solve sub-problems and use the solutions. Moreover, program design requires the definition of new abstractions which, in general, can be regarded a particularly demanding cognitive task (Blackwell, 2001). The unfamiliarity with the methods in this area often leads the student to stare at the blank computer screen with no idea of what to do next.

A third difficulty is caused by the strict syntax and semantics of programming languages. The notation of programming languages seems often complex for beginning programmers. Moreover, they are usually unfamiliar with any tasks that require similar meticulous and careful attention to detail as the use of programming language does. No errors in syntax and semantics of programs are tolerated.

Fourth, programming requires familiarity with a set of specific tools — editor, interpreter or compiler, debugger, and a manual browsing tool — and appropriate work practices. An example is iterative development style — composed of editing, compilation, and testing steps — that a student should learn to use in a proper, goal directed manner, instead of resorting to ineffective trial-and-error style (so-called bricolage (Ben-Ari, 1998)).

Programming cannot be learned without some level of parallel progress in all these areas, even if the mastery in the skill only comes after years of practice.

PBL has an important role in learning new concepts and mechanisms. A properly designed case makes the student to wonder about the concepts and raises questions about them. Subsequent self study thus falls in fertile ground. In addition, the discussion in the closing session can aid the conceptual learning by revealing different interpretations and viewpoints to the concepts. It can be instructive to see, what kinds of difficulties or misunderstandings others have had with the concepts.

Concept maps and essays can be effective tools in further clarifying the meaning of the concepts and their relations with each other. It should be stressed that concepts that have a central role in some dynamic mechanisms of a programming language require active working with these concepts. For this, different kinds of programming exercises are indispensable.

Problem solving and program design skills need active training. The use of design cases in PBL sessions has turned out to be a successful way to do that. Students can practice analysis and design skills in several different cases. Working in a PBL group helps the design process considerably. There are more perspectives, which typically aids in the analysis of the problem. PBL groups do not get stuck to one unproductive point-of-view as easily as individuals do. Moreover, when students try to explain the problem or proposed designs to each other, they naturally develop more abstractions. This generally improves the design of the program. As a result of all of this, the students feel that design work is more fun in a group than when done alone.

At the first glance, PBL cases do not appear suitable for learning the syntax and semantics of a language. However, we have used fairly successful cases to introduce the syntax of HTML and Java. Both of these cases have been similar: students are handed a page of code and something resulting from the interpretation of the code (for instance, a picture of a rendered page or the output of a program). After the initial confusion, the PBL group usually starts to look the code more closely and is able to form many hypotheses of what different constructs might mean. This has served as a good way to motivate the subsequent learning of the syntax and interpretation of language constructs. However, in order to reveal the strictness of the form and interpretation of a program to a student, active working with a computer is required.

Finally, learning to use programming tools or to apply effective programming practices requires exercises and demonstrations. PBL has little role in these areas.

6. SUMMARY

In this paper we have presented an implementation of an introductory programming course in Java using the Problem-Based Learning approach, especially the seven step method. During the years 1999 – 2003 we have carried out the course five times, and tested various types of PBL cases and combinations of PBL and other learning methods. We have tried to identify the best practices to implement a PBL course in programming.

We have recognized some areas in the programming skill that are well trained using the PBL method and other areas where complementary methods are needed. PBL sessions are an effective way to create motivation to learn abstract concepts. The discussion in the closing session of a PBL case is useful to deepen the understanding of the meaning of the concepts. The use of concept maps, either in the PBL session or in the individual learning, clarifies the relations between the concepts. Programming exercises are indispensable in learning the practical use of the concepts. The exercises are also needed to learn the use of programming tools and work practices.

PBL sessions are fruitful in learning problem solving and programming design. In the PBL group, the problem is analyzed from different perspectives. When students try to explain the problem or proposed designs to each other, they naturally develop more abstractions. This generally improves the design of the program.

The PBL method has other advantages not related to programming. First, students learn working in groups, creative methods such as brainstorming, and planning of their work. Second, they train their communication and argumentation skills. Third, they get support from their peers that helps relieve the anxiety caused by difficult topics. Fourth, they learn to search for information from different sources. Finally, they learn to take more responsibility of their studies. This seems to carry over to subsequent courses in their studies.

NOTES

1. A concept map is a simple diagrammatic representation of a set of concepts and named relations between these concepts (Novak & Gowin, 1984).
2. This emphasis may be different in less technical areas where different (possibly conflicting) viewpoints exist.

REFERENCES

ACM/IEEE Curriculum 2001 Task Force. (2001). *Computing Curricula 2001*, Computer Science.

Ben-Ari, M. (1998). Constructivism in computer science education. *SIGCSE98*, 257 – 261.

Blackwell, A. (2001). See what you need: Helping end-users to build abstractions. *Journal of Visual Languages and Computing*, *12*, 475 – 499.

Fekete, A., Greening, T., & Kingston, J. (1998). Conveying technical content in a curriculum using problem-based learning. In *Third Australasian Conference of Computer Science Education* (pp. 198 – 202), Brisbane, Australia.

Greening, T., Kay, J., Kingston, J., & Crawford, K. (1997). Results of a PBL trial in first-year computer science. In *Second Australasian Conference of Computer Science Education* (pp. 201 – 206), Melbourne, Australia.

Kay, J., Barg, M., Fekete, A., Greening, T., Hollands, O., Kingston, J.H., & Crawford, K. (2000). Problem-based learning for foundation computer science courses. *Computer Science Education*, *10(2)*, 109 – 128.

Lambrix, P. & Kamkar, M. (1998). Computer science as an integrated part of engineering education. In *Third ACM SIGCSE/SIGCUE Conference on Integrating Technology into Computer Science Education* (pp. 153 – 156), Dublin, Ireland.

Norman, G. & Schmidt, H. (1992). The psychological basis of problem-based learning: A review of the evidence. *Academic Medicine*, *67(9)*, 557 – 565.

Novak, J. & Gowin, D. (1984). *Learning how to learn*. Cambridge: Cambridge University Press.

Schmidt, H. (1983). Problem-based learning: Rational and description. *Medical Education*, *17*, 11 – 16.

Thagard, P. (1990). Concepts and conceptual change. *Synthese*, *82*, 255 – 274.

Ventura, P. (2003). *On the Origins of Programmers: Identifying Predictors of Success for an Object First CS1*. Doctoral thesis, State University of New York at Buffalo.

APPENDIX

PBL cases

Each year we have ten cases in the course. The subject areas of the cases are mostly same in each year, but the case may vary. Below we list the subject areas and cases that we have used. In addition, we describe some difficulties that the students have had with the cases and the way our tutors have helped the students in the cases.

Case 1: HTML. We present a simple HTML page rendered by a web browser and the corresponding HTML source file. There are two reasons for having this sort of case in the course. First, the students write all documents in the course in HTML. Second, HTML serves as a simple example of a structured language resembling programming languages. This case has been rather easy for the students.

Case 2: Introducing the concepts of an object and a class. We have used two different cases, the Plato's theory of eternal forms, presented in section 3.3 Examples of Cases, and a hamburger restaurant case. In the latter, the students are asked to develop a model for analyzing the operation of a hamburger restaurant. In the model, they should identify objects and classes present in the problem. If the students find this task difficult, the tutor may ask them to think what kinds of concepts are related to hamburger restaurants. If the students describe very abstract concepts, the tutor may ask them to list things that they have seen in hamburger restaurants. Sometimes the students mistake subclasses for objects — for instance they may think that a hamburger is a class and a cheese burger is the instance of this class. The learning goals established by the students are related to objects and classes and their presentation in Java.

Case 3: Java program code. We give the students the listing of some simple Java classes (about 60 lines of code in total) and the corresponding program execution. In case 2, the students learned the concepts of an object and a class. The aim here is to make them more familiar with the structure of Java programs and the exact Java syntax, to introduce constructors, methods, and variables, and to present the idea of program compilation and execution. The students typically detect some connections between the program code and the program execution, but find the program syntax incomprehensible. The learning goals are mostly related to Java syntax.

Case 4: Designing the control logic. We have used two different cases. In the first variant, the students should design the control logic for driving an elevator in a building. The students should first identify the objects and classes needed and then try to somehow describe the program logic. They may use some kind of pseudo code or draw a flow chart. In the second variant, the students are asked to design a control logic for a robot in a maze. They are presented pictures of different kinds of mazes. To make the task easier, they are told that the robot can turn itself left or right and step forward if it is possible. The students should first identify objects and classes needed for presenting the maze and then describe the control logic. Overall, the students seem to find the robot case more interesting than the elevator case. Although the design tasks are not easy, the students get fairly good algorithms. They usually take for their learning goals to be able to define the objects and classes needed and the control logic in Java.

Case 5: Collections. The students are asked to design a way to present data in a WWW bookstore. They need to present the available products, and the customers and their shopping carts. Obviously, collections of objects are needed here. The students sometimes seem to have difficulties in distinguishing between classes and collections, since objects somehow "belong" to both classes and collections. The tutor may have to point out their differences. The learning goals are related to presenting collections in Java.

Case 6: Hierarchic object structure and recursion. The students should design the class structure for a computer file system. The case was presented in Section 2.4.

Case 7: File input and output. We have used two different cases. In the first case, the students consider how the state of a computer game can be saved in a file and later be restored from the file. In the second case, the problem is to list all URLs in an HTML file. A difficulty in this case is that Java has a rather complicated IO-class hierarchy and there are different classes for handling streams consisting of eight bit bytes and 16 bit Unicode characters. This is not, however, a problem in the opening session, since the students approach the problem in a higher level of abstraction.

Case 8: Graphical user interface components and layout. As the case we have used the digital photo album. The students should design the appearance of the user interface and find out how they can implement it in Java Swing.

Case 9: Event handling and dynamical behavior in a graphical user interface. The students should design a ship sinking game and find out how they can implement it in Java.

Case 10: 2D graphics in Java. The students consider the differences between two kinds of interactive 2D drawing programs. In the first one, the shapes drawn cannot be moved or modified, whereas in the second one the user can move and modify the shapes. The students should find out, how each kind of program can be implemented in Java.

Programming Exercises

There are five programming exercises and for each exercise the students have two weeks time to complete it. The technical theme of an exercise is related to that of the recent PBL cases. For example, in the first exercise the theme can be simple Java classes (related to cases 2 and 3), in the second exercise the control structure (related to cases 3 and 4), and in the last exercise a graphical user interface (cases 8 and 9).

In some course versions the application area has been different in each exercise, but in other versions the same application area has been used in most exercises. For instance, in autumn 2002, the application area of the first four exercises was an interactive text based adventure game.

Program design is trained in many PBL cases. On the other hand, in the exercises the students do not design the class structure and the methods needed. The classes, method signatures and the requirements for the method behavior are described accurately in the exercises. With this sort of guidance, the students can implement rather advanced programs and see examples of good program designs.