Linking Theory, Practice and System-Level Perception: Using a PBL Approach in an Operating Systems Course

Moshe Pelleh, Bruria Haberman, and Tammy Rosenthal
Computer Science Department, Holon Institute of Technology, Holon, Israel

mpelleh@yahoo.com; bruria.haberman@weizmann.ac.il; tammy.ros@gmail.com

John English
School of Computing, Mathematics and Information Sciences, University of Brighton, Brighton, UK

je@brighton.ac.uk

Abstract

Courses on Operating Systems (OS) are essential in computer science education. The topic provides the students with an excellent opportunity to experience the interplay between theory and practice. Specifically, a project-based-learning (PBL) instructional design for an OS course can provide the students with opportunities to engage in practical projects. The PBL approach enables students to take part in learning activities which are essential for grasping underlying theoretical concepts, linking theory, practice and system-level perception. In this paper we present our experience in teaching an operating systems course with a continuing evolving project using a PBL approach. The findings of a preliminary assessment indicated a highly positive attitude on the part of the students towards the PBL approach used in the course as well as towards the qualitative evaluation method that was used to assess their achievements.

Keywords: Operating systems, project-based learning, system-level perspective, professional practice.

Introduction

In the modern world practical computer science integrates scientific subjects and technological applications. The dynamics of the field of computing along with the strengthening of ties between the various sciences and technology requires suitable manpower to be trained. This training should be carried out within the framework of the various technological systems integrated in the computer, together with a broad specialization in the computer sciences. The Holon Institute of Technology (HIT) meets this need and offers

a suitable academic track in computer science. HIT is an institution of higher learning that specializes in the scientific, professional, social and cultural aspects of advanced technology. The institute is grants degrees in various domains, one of which is a BSc in Computer Science (CS). Since the development of modern high-tech industries requires the training of highly skilled personnel, some of HIT's primary educational goals are [http://www.hit.ac.il/about/profile.asp; retrieved November 1 2007]:

- *Aspiring to excellence in instruction and research at the cutting edge of technological development by identifying new spheres of knowledge, emerging interdisciplinary subjects and innovative instructional technologies.*
- *Aspiring to the realization of students' intellectual and professional potential and training personnel for careers in a constantly and rapidly changing technological world; providing opportunities to maximize students' self-fulfillment.*
- *Fostering close ties with science-based industries for cooperation in research and, applied technology to promote scientific-technological education.*

In this paper we present our experience in teaching one specific course of the undergraduate computer science program in accordance to the above-mentioned general goals. First, we describe the general framework of the Computer Science program and specifically present the outline of the Operating Systems (OS) course. Next we present the instructional design of the OS course and its underlying pedagogical approach. Finally, we present the findings of a preliminary assessment of the course's implementation.

The Undergraduate Computer Science Program in HIT

The curriculum is designed to provide a B.Sc. in computer science and provides the basic theoretical and applied knowledge necessary for graduates to be able to work in a variety of modern high-tech environments. Specifically, we believe that graduates should gain an integrative holistic knowledge of the discipline and possess: (a) a basic theoretical background in science (i.e., mathematics and physics); (b) a solid theoretical background in CS; and (c) a deep knowledge in some advanced specialization track or a breadth-oriented knowledge of variety of advanced topics (as illustrated in Table 1). The curriculum supports a method integrating multidisciplinary study and technological skills with technology, hardware and software.

The student population consists of two groups: (a) students who are capable of devoting most of their time for studying, who attend a day tuition track; and (b) students who combine study and work and attend a special flexible evening track. Most of these students work already in high-tech industry. As a result, the student population is diverse with respect to previous knowledge in computing as well as in practical experience and acquaintance with the "real world" industrial situations.

The Structure of the Program

The 3 year, 147 hour program offers a variety of theoretical and application-based courses related to fundamentals as well as to advanced computing topics. The main topics studied are algorithmics (problem solving, algorithm planning and analysis), an understanding of the structure and working of the computer, programming languages and the uses of computer science in various domains. The structure of the program is illustrated in Table 1. The student may choose a combination of elective courses which are necessary to acquire deep knowledge in some advanced specialization track, or decide to take a variety of elective courses to gain a broad knowledge of advanced topics (total: 7 elective courses, 28 hours).

Table 1. The structure of the program (Total: 147 hours)

<table>
<tr><td colspan="3">Mandatory science courses (math, physics) [8 courses, 35 hours]</td></tr>
<tr><td colspan="3">Mandatory computer science courses [17 courses, 78 hours]</td></tr>
<tr><td colspan="3">Elective computer science courses in various topics
or courses related to a Specialization Track
[total: 7 courses, 28 hours]</td></tr>
<tr><td colspan="3">Specialization Track</td></tr>
<tr><td>Embedded Systems</td><td>VLSI Design</td><td>Artificial Intelligence</td></tr>
<tr><td colspan="3">General courses [3 courses, 6 hours]</td></tr>
</table>

Objectives and Pedagogic Approach

The program has two main objectives which follow the guidelines of CC 2001 (ACM/IEEE Joint Task Force on Computing Curricula, 2001): (1) to enable students to gain an integrative knowledge of computer science, and (2) to prepare students to achieve a full-fledged membership in the computing community of practice. The program is designed to address its goals in the following dimensions:

(1) Perception: Graduates must develop a system-level perspective, understanding the interplay between theory and practice, and appreciate the value of good engineering design.

(2) Performance: Graduates must develop their cognitive capabilities and practical skills, and be capable of combining scientific thinking with problem-solving skills. They should learn to integrate theory and practice and recognize the importance of abstraction.

(3) Professional practice: Graduates should be exposed to behavioral norms and industrial standards, and possess the aptitude for life-long self-study of new technologies.

The instructional design is directed towards achieving the above goals as follows:

Constructing integrative knowledge: Although the body of computing knowledge is actually an integrated whole, the students might not recognize the linkages between concepts learned within each course, and even more so the concepts learned in separate courses. Hence, the instructional design should support the construction of integrative knowledge by linking concepts. The HIT CS department tries to address these goals by designing a variety of suitable learning activities. The activities develop gradually, from tasks at a "paper and pencil" level, exercises in the laboratory, and developing an application (a mini-project or a comprehensive project) as a final assignment for the course.

Preparing for professional practice: According to the socio-cultural approach (Lave & Wenger, 1991), preparing students to become professionals should be dealt with during their studies. The HIT CS department tries to achieve this goal by:

(a) Active participation of the community's representatives, which combines a theoretical background with a rich practical experience. In the context of this paper, the first author (the lecturer of the Operating Systems course presented here) fulfills these criteria.

(b) The program is supported by advanced laboratories that closely follow industrial practice and standards, each of which is dedicated to a specific domain. The laboratories are equipped with the most innovative and advanced facilities that enable performing a variety of practical exercises

and projects addressing topics in hardware, software, and various applications. Use of these laboratories enables effective experimental and integrative learning of the program's subject areas.

(c) Incorporating real-world elements into the curriculum. This enables the effective learning of concepts and practical skills (ACM/IEEE Joint Task Force on Computing Curricula, 2004). We try as much as possible to integrate such activities in courses that combine theoretical and practical aspects, one of which is the Operating Systems course, which we refer to in this paper.

In the following section we describe the problem-based learning (PBL) pedagogic approach, which is widely applied in teaching computer science in our department; later, we specifically focus on the use of this approach in teaching the Operating Systems course.

Project-Based Learning (PBL)

The academic CS community believes that the role of projects in the curriculum is of great importance. Final course projects are recognized as valuable for enhancing student knowledge, and especially for acquiring a system-level approach. Since the nature of work within most modern organizations is moving from individual assignments to team and project-based activities, it is important to integrate projects into the learning process to simulate real world professional situations. However, the integration of such course projects should be carefully designed, not merely as "post-course" projects (CC 2001). In recent years, the potential of project-based learning has been realized (Fincher & Petre, 1998; Matsuura, 2006). Project-based learning (PBL) is a constructivist pedagogy that intends to bring about deep learning by allowing learners to use an inquiry based approach to engage with complex issues that require students to investigate in order to understand (http://en.wikipedia.org/wiki/Project-based_learning). Constructivism describes *how learning should happen*, regardless of whether learners are using their experiences to understand a lecture or attempting to design an artifact. The theory of constructivism suggests that in both cases, learners construct knowledge (Ben-Ari, 2001; von Glasersfeld, 1989). Constructivism is often associated with pedagogic approaches that promote learning by doing, such as: active learning (Bonwell & Eison, 1991) and cooperative learning (Bossert, 1988). Project-based learning shifts the emphasis away from a traditional teacher-centered approach to student-centered teaching and active and cooperative learning. Project-based learning is very different from traditional learning in which the student deals mainly with short exercises and well-defined "local" problem solving tasks. Project work on the other hand requires the student to develop an entire system which is assembled from "small pieces". Actually, the PBL approach emphasizes on long-term, student-centered learning activities, and on students' own artifact construction to represent what is being learned. In a project-based framework students must collaborate with peers, organize their own work and manage their own time.

Based on the above considerations, the PBL approach enables students to construct knowledge and to enhance their cognitive and reflective skills; it encourages students to become creative and independent learners, and enables them to gain real experience as project developers (Fincher, Petre, & Clark, 2001; Holcombe, Stratton, Fincher, & Griffiths, 1998). The approach can be proved to be a very suitable approach to prepare undergraduates for their professional life, when cooperative working will form a major part of their work duties (Wegner, 1998).

The Operating Systems Course

An operating system (OS) is the software that manages the sharing of the resources of a computer and provides programmers with an interface used to access those resources. An operating system processes system data and user input, and responds by allocating and managing tasks and internal system resources as a service to users and programs of the system. At the foundation of all system software, an operating system performs basic tasks such as controlling and allocating

memory, prioritizing system requests, controlling input and output devices, facilitating computer networking and managing files. Most operating systems come with an application that provides an interface for managing the operating system. The operating system forms a platform for other software (http://en.wikipedia.org/wiki/Operating_system).

Operating Systems (OS) is a fundamental topic in computer science education. Learning the subject provides the students with an excellent opportunity to experience the interplay between theory and practice. In our program, the course is taught in the last year after courses in Computer Structure, Assembly Language and Systems Programming. An outline of the course content is as follows:

Introduction: Computer system organization; operating system structure; simple batch systems; resident monitors; off-line processing; spooling; random access; time-sharing systems; multiprogramming batch systems; loosely coupled systems; tightly coupled systems; real time systems.

Computer structure and operation: Computer structure; computer system operation; interrupt handling; input/output structures; memory structure; other hardware features.

Operating system structure and operation: operating system structure; operating system services; virtual machines; virtual memory; virtual I/O.

Process management: Process concepts; process states; process control blocks; operations on processes; interprocess communication; context switching.

Threads: Overview; multithreading models; thread libraries.

CPU Scheduling: Scheduling criteria; scheduling algorithms; dispatching; the multiprocessing case; real time systems.

Advanced scheduling: Priority; task models; fixed priority model; task level dynamic priority; job level dynamic priority; RM algorithm; EDF algorithm; LST algorithm; round Robin scheduler; soft and hard real time; priority inheritance.

Process synchronization: The critical section problem; Peterson's solution; synchronization hardware; semaphores.

Deadlocks: Deadlock characterization; deadlock prevention and avoidance; deadlock detection; recovery from Deadlock; the priority ceiling protocol.

Main memory: Address binding; dynamic loading; dynamic linking; overlays; logical address space versus physical address space; swapping; paging; segmentation; MMU and page tables.

Virtual memory: Demand paging; page replacement; allocation of frames; thrashing; memory mapped files; memory mapped I/O.

File management and I/O systems: File structure; access methods; directory structure; free space management; performance; I/O hardware; OS I/O features; OS I/O calls.

Protection and security: Protection; protection domain; access matrix; security; password; viruses; worms.

The instructional method is based on lectures combined with exercises in the laboratory. A set of theoretical books are recommended for reading (Andrew, Tanenbaum, & Woodhull, 2006; Liu, 2000; Silberschatz, 2005). The exercises are planned so that students implement the concepts learnt in the lectures, and their implementations also serve as building blocks for a large-scale project.

The project is implemented in the Embedded Real-Time Systems laboratory, whose design closely follows industry standards. It facilitates the research, study and development of software

for real-time embedded systems as well as other industrial software. The laboratory is equipped with unique hardware and software that enables cross-platform (host-to-target) software development and supports courses such as Real-Time Systems, Computer Networks, Operating Systems, and others. The students working in the laboratory have the opportunity to become acquainted with work which is typical of an industrial environment.

A Project in an Operating Systems Course

The computer science academic community regards team project work as essential (Fincher & Petre, 1998). In particular, projects in operating systems courses are reported as valuable for the students' learning processes (Guzman & Bobbie, 2007; Krishnamoorthy, 2002; Ramakrishnan & Lancaster, 1993; Sharma, 2007). In this section we present our PBL approach for incorporating comprehensive project work into the learning process of the Operating Systems course. Our approach is consistent with the educational philosophy of CDIO™ INITIATIVE which is an innovative educational framework for producing the next generation of engineers that provides students with an education stressing engineering fundamentals set in the context of Conceiving-Designing-Implementing-Operating real-world systems and products (http://www.cdio.org).

Work on the projects takes place during the whole course. The students develop the project from scratch in parallel to learning the theoretical material, implementing it in gradual stages. The project can be a current project chosen by the student with strong usage of OS elements, which must be approved and supervised by the lecturer, or our basic project, defined by the lecturer and described later in this paper.

The Rationale behind a Project in an OS Course

In this course we plan and develop a project during the whole semester, starting from the very beginning of the course. At the end of the semester the student presents his or her project and defends it. Each week we add characteristics and refine the requirements of the expected product following the progress of our lectures until we get a complete product (an operating system) at the semester's end. During the project we try to simulate an industrial environment. The lecturer plays the role of a client for the functional requirements and acceptance testing as well as the role of the project manager for planning the timetable and high-level design. The students are grouped in teams of two, so if someone is missing one day, the other one can continue in order to keep to the timetable derived from the lectures. On the other hand, a team of two is small enough to avoid passive participation. Furthermore, in a real workplace, work is generally done by teams and here we imitate this situation. The course, by its nature, is a practical course with topics and algorithms suitable for practical implementation. Working during the semester in this setting helps the students to understand the theme better than taking a short one-off examination. Moreover, a project gives a better feeling of reality and a better preparation for work. Learning to create a project has four basic dimensions:

1. Depth and width: deep understanding of a variety of complex situations.
2. Independency and creativity: the ability to work independently, to propose solutions to problems, and to invent new applications.
3. Real world sensitivity: working in industrial conditions with typical tools, working hours, partners, etc.
4. Organization and engineering: using methods, standards and plans for the development process.

On the other hand, preparing for a written examination will only address the first two dimensions.

The OS Project Plan

The project process is planned to cover most of the important topics of the course in adequate time. This is so that students can learn and assimilate the concepts without impacting on other courses in the same semester. Several aspects related to planning the project are described below.

Project span:

The project includes about 30 topics in programming that cover most of the course program. Supposing an average of 25 lines of C per topic, we get about 750 lines of C code, which will produce, after compilation, about 2000 assembly language instructions. Supposing that an assembly language instruction occupies an average of 4 bytes, we get about 8K of machine code.

Project time consumption:

The productivity of a student is estimated as 10 lines of C code per hour, hence the investment in the project is about 80 hours. Adding 20 hours for planning, organization, debugging and documentation, we get about 100 working hours per project.

Student's work load:

A development team is comprised of two students, so the load per student is about 50 hours per semester. Since we have about 12 weeks in a semester, this corresponds to about 4 working hours per student each week.

The students have an hour a week of guided exercises in the course, which are used for the project. Three hours a week are left for independent work which is regarded as homework.

Requirements:

Requirements are added or deleted according to weekly workload feedbacks. Since there are a few common algorithms used to implement each topic, we require implementations of one of them, all of them, or a few of them according to their importance and depending on the workload.

The OS Project's Tasks

The whole product is assembled from the following components, which are given as lab and homework team-assignments:

Process management: a process, defined as a unit of execution on the computer, should be created, deleted, identified, handled, executed and often made to communicate with other processes. To achieve these goals the operating system contains the following: *process creation, process termination, process control block, ready queue and various I/O device queues, process states, context switching, forking a process and interprocess communication.*

Here the student must plan and implement all the topics listed above in order to proceed.

CPU Scheduling: there are a variety of algorithms for scheduling processes on the CPU. For example: *First come first served, Shortest-Job-First (SJF) Scheduling (Non-Preemptive SJF, Preemptive SJF), Fixed Priority scheduling, Round Robin scheduling, Multi level queue scheduling, Earliest Dead line First, Least Slack Time first.*

The students should choose at least one algorithm to implement for their operating systems. They can add Priority inversion or priority inheritance feature (to score more points). They can use Round Robin scheduling in an equal priority task group managed by Fixed Priority scheduling. They can choose to implement Multi level queue scheduling or Least Slack Time first instead of

First Come First Served (which scores minimum points). In addition, they can provide the user with a menu to choose between algorithms.

Process synchronization: there are few mechanisms for process synchronization: *Synchronization Hardware, Peterson's Solution, Semaphores, and Semaphores with no busy waiting.* The student should implement one of these synchronization mechanisms. A better solution will score more points.

Deadlocks: deadlocks are situations where processes are blocking each other forever. Such situations should be avoided, and we have some methods to avoid and handle deadlocks. The most common are: *Resource-Allocation Graph Algorithm, Banker's Algorithm (Safety Algorithm), Priority ceiling protocol.*

The student should implement at least one of the above listed algorithms. Implementing more algorithms scores more points.

Memory Management: the following commonly used mechanisms are covered: *Hierarchical Paging, Hashed Page Tables, Inverted Page Tables.*

The student should implement at least one of the above.

Virtual Memory: a few page replacement policies are covered. In the project the students implement the LRU Approximation Algorithm – Second chance.

Evaluation of a Project

The criteria for evaluating a project and for assigning grades are based on the following factors:

1. Quantitative: how many tasks are implemented in the project.
2. Qualitative: the human interface, efficiency, options like monitor, warnings, etc.
3. Depth – How the tasks were implemented.

The following are examples for evaluating the "Depth" factor:

(a) For the fixed priority scheduling, in the case where we have few processes with the same priority, we can treat them on a First Come First Served basis. A student who treats them under the Round Robin approach will score more points.

(b) Students who use the Test-And-Set instruction to implement process synchronization in a critical section will score more points than students who use Peterson's Algorithm or who disable interrupts.

Here is an implementation using the Test-And-Set assembler command:

```
L:      TAS     X       ;   Test And Set X to 1
        BNZ  L          ;    if x was not 0 go to L
        command   ;   critical section
        command   ;   critical section
        command   ;   critical section
        MOVE 0,X  ;   Set X to 0
```

(c) Another example relates to implementing a queue for use as a mailbox between 2 processes: using the circular queue (and avoiding the need of a semaphore) will score more points than an approach using a critical section.

Here is an implementation of a circular queue in pseudo code:

```
Int n = size_of_mailbox
Int head = 0, tail = 0
Char mailbox [n]
void receive() {
        if (head ≠ tail) {
                data = mailbox(head);
                head = (head+1) mod n;
        }
}
void send() {
        if ((tail +1) mod n ≠ head ) {
                mailbox(tail ) = data ;
                tail = (tail+1) mod n;
        }
}
```

Each week we check each team's newly added component (which was completed as homework, its integration with earlier components, and improvements of earlier components (from previous weeks). This is used to award each team its weekly grade. This weekly grade can never be modified, although if a project is improved later, the team will be awarded additional points in a later weekly grade. The final grade in the course is the sum of the weekly grades. Since weekly presentation cannot be delayed and a grade cannot be modified, students must stick to the timetable and must do their best in the time available, which is a good habit for future engineers.

Student Feedback

In order to get the students' feedback on the Operating Systems course and its project, a questionnaire was distributed at the last week of the semester.

The students were asked to fill the form anonymously. A total of fifty-four students filled in the forms. Twenty-three students out of this total of fifty-four students also answered the open questions. The questions and results (average grade) are presented in Table 2. Some of the responses to the open questions are also presented.

Following are some quotations from the open responses students gave on question 10, "What are the most significant factors for you in the project?"

- *"The guidance, focusing and team work."*
- *"Having the option to work independently and get more detailed knowledge of selected topics."*
- *"Simulation of the algorithms taught in the course helps a lot to grasp the material."*
- *"Better understanding of the topic of memory management in OS."*
- *"Implementing memory management and understanding OS processes."*

Table 2: The Results of Students' Feedback Forms (N=54)

	Question	Average Mark	Percentage of Stud.
1	How effective were the given instructions and the guidelines before and during your work on the project? (Scale: 1-5. 1: Not effective at all; 5: Very effective).	4.648	
2	Did the project match the material studied in class? (Scale: 1-5. 1: Did not match; 5: Matched very well).	4.833	
3	Did the implementation of the project help you understand the course material better? (Scale: 1-5. 1: Did not help at all; 5: Very helpful).	4.833	
4	Grade the workload of the course and the project as a whole. (Scale: 1-5. 1: High workload ; 5: Low workload).	3.463	
5	Would you rather have in the OS course: • A final exam (mark: 1), or • A final project (mark: 5)?		13% 87%
6	In computer science courses, do you prefer to have: • A final exam (mark: 1), • A project (mark: 5), or • Does it depend on the type of course (mark: 3)?		33% 63% 2 students
7	Did you enjoy the course? (Scale: 1-5. 1: Did not enjoy at all; 5: Enjoyed very much).	4.648	
8	Was the course useful for you? (Scale: 1-5. 1: Not useful; 5: Very useful).	4.630	
9	Would you recommend to your friends to take this course? (Scale: 1-No. 5-Yes).	4.722	
Open Questions			
10	What are the most significant factors of the project (In other words, what are the factors that contributed most to your understanding of the course material)?		
11	Additional comments:		

The feedback as a whole was very good. Most students (42 out of 54) clearly preferred to have a project rather than a final exam in OS and gave a grade of 5 in question 5. A few students interpreted the question as requiring a response on a scale from 1 to 5 rather than a dichotomy and therefore responded with 3 (one student) or 4 (4 students) as their scale. Nevertheless it is clear that these responses also tend to favor a project. Interestingly, the students were less determined when answering a similar question (question 6) regarding any course in the program; only 66% would prefer a project in general.

In the open questions, all but one of the answers were positive about the course. The negative response was that “working at the same time as the presentation took place would have made it easier to grasp the material.”

The factors described above were repeated by many students. For example, four students mentioned that the project helped them to understand the topic of memory management better. Seven students mentioned as the most significant factor that the project helped them to grasp the algorithms taught in the course.

Late Post Interviews

A few interviews were conducted four months later, to gather the students' views after a period for reflection. Here we present some of the students' reflections.

Students A, B, and C belong to the flexible tuition track and combine their studies with work in high-tech industry. The students reported on their practical industrial experience.

Student A pointed out that the most important factor in the project work was its systematic and disciplined development: "one must consistently follow a process from the beginning to the very end". In his opinion, it is difficult to practice a disciplined methodology when studying theory *per se*. Moreover, the project work requires ongoing learning during the whole semester (in contrast to studying for a final exam). The project links theory and practice; the material becomes more tangible and perceptible when the theoretical concepts are implemented. The student indicated one specific shortcoming of the project: one has the opportunity to avoid coping with topics and assignments which seem difficult; naturally, students focus on topics that they understand better as well as on actions that they can complete. To avoid this "getting out of doing something", a strict definition of requirements of the project needs to be explicitly stated by the instructor.

Student B was pleased with the opportunity to perform a project related to his field work. Actually, the project assignment motivated the student to complete a practical project in his work place, which for various reasons had been postponed until then. Interestingly, the student reflected on the project work from a pedagogical point of view and reported on processes that occurred in class during project work. For example, he stated that for inexperienced students the project served as an example of typical work practices in industry. The experienced students served as role models for the inexperienced students and demonstrated to them their practical knowledge. His impression was that the project work enabled learning at different levels at and different paces.

The third student (C) stated that a mentoring role model is more important than the "standard instructor" model since it resembles the "real industrial world". The main benefit from the project work for student C was that it made her regard working practices in a different way; in particular, the need for accurate specifications as one of the main issues that one should pay attention to when developing a project. Actually, the OS project seemed to student C as fair, a good thing to do, and a contribution to a lifelong learning habit.

Concluding Remarks

In this paper we have shown how PBL principles have been applied to the teaching of an Operating Systems course at undergraduate level. The course provided the students with an excellent opportunity to experience the interplay between theory and practice. Actually, the PBL educational approach could be used on any computer science course which combines theoretical and practical aspects. We have presented how the concept of a “project” can be embedded in a pedagogical framework, combining traditional and modern didactical methods. The projects were designed to involve gradually increasing levels of difficulty; as a result, the students approached the

complexity of the system incrementally. Using this approach we enabled students to solve practical problems in a realistic environment that simulates "real world" industrial scenarios. The findings of our preliminary assessment indicated a highly positive attitude on the part of the students towards the teaching and learning approach employed in the course as well as towards the qualitative evaluation method that was used to assess their achievements. The overall course assessment demonstrates the pedagogical value and effectiveness of the PBL model as a means of improving students' personal, collaborative and communication skills. The course resulted in students changing many of their attitudes towards learning and assessment and they started to adopt a more professional way of thinking. In order to obtain a more concise in-depth view we plan to conduct more interviews with students, during further implementation of the course in different stages of the project development processes.

References

ACM/IEEE Joint Task Force on Computing Curricula. (2001). Final Report, December 2001.

ACM/IEEE Joint Task Force on Computing Curricula. (2004). Software Engineering 2004 Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering, A Volume of the Computing Curricula Series, August 2004.

Andrew S., Tanenbaum, A., & Woodhull, S. (2006). *Operating systems design and implementation* (3rd ed.). Prentice Hall.

Ben-Ari, M. (2001). Constructivism in computer science education, *Journal of Computers in Mathematics and Science Teaching*, *20*(1), 45-73.

Bonwell, C., & Eison, J. (1991). *Active learning: Creating excitement in the classroom.* AEHE-ERIC Higher Education Report No.1. Washington, DC: Jossey-Bass.

Bossert, S.T. (1988). Cooperative activities in the classroom. *Review of Educational Research*, *15*, 225-250.

Fincher, S., & Petre, M. (1998). Project-based learning practices in computer science education. In proceedings of: *IEEE Frontiers in Education Conference*, Tempe, Arizona, USA, November, 1185-1191.

Fincher, S., Petre, M., & Clark, M. (Eds.). (2001) *Computer science project work principles and pragmatics*. London: Springer-Verlag.

Holcombe, M., Stratton, A, Fincher, S., & Griffiths, G. (Eds.). (1998). Projects in the computing curriculum. *Proceedings of the Project 98 Workshop*. London: Springer-Verlag.

Guzman, J. C., & Bobbie, P. O. (2007). Hands-on operating systems made easy. *JCCSC, 22*(4), 145-151.

Krishnamoorthy, S. (2002). An experience teaching operating systems course with a programming project. *JCCSC, 17*(6), 25-38.

Lave, J., & Wenger, E. (1991) *Situated learning: Legitimate peripheral participation*. Cambridge, UK: Cambridge University Press.

Liu, J. W. S. (2000). *Real-time systems.* Prentice Hall.

Matsuura, S. (2006). An evaluation method of project based learning on software development experiment. In proceedings of *SIGCSE'06,* Houston, Texas, USA, March, 163-167.

Ramakrishnan, S., & Lancaster, A. M. (1993). Operating system projects: Linking theory, practice and use. *ACM SIGCSE Bulletin*, *25*(1), 256-260.

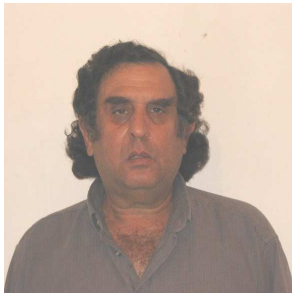
Sharma, O.P. (2007). Enhancing operating system course using a comprehensive project: Decades of experience outlined. *JCCSC, 22*(3), 206-213.

Silberschatz, A., Galvin, V. P., & Gagne, G. (2005). *Operating system concepts* (7th ed.). John Wiley & Sons.

von Glasersfeld, E. (1989). Cognition, construction of knowledge, and teaching. *Synthese, 80,* 121–140.

Wegner, E. (1998), *Communities of practice: Learning, meaning and identity*. UK: Cambridge University Press.

Biographies



After completing his doctoral studies at the University of Pisa, **Moshe Pelleh** developed several advanced projects in the communication and aerospace industries involving real time embedded systems. In the last years Dr. Pelleh has been a senior lecturer in Holon Institute of Technology, teaching courses in Network Communications, Operating Systems, Real Time Systems and Embedded Systems. He is the founder and head of the Real Time Embedded Systems Laboratory. His research is centred in the field of Real Time Embedded Systems.

Bruria Haberman received her Ph.D. degree in Science Teaching from the Weizmann Institute of Science. She is currently a faculty member in the Department of Computer Science in the Holon Institute of Technology, teaching Logic Programming, Data Base Systems and Expert Systems. She is also a member of the computer science team in the Davidson Institute of Science Education in the Weizmann Institute of Science, where she leads the Computer Science, Academia & Industry educational program for talented high school students and their teachers. She is also a leading member of Machshava, the Israeli National Center for computer science teachers. She has developed learning materials for high school level in the areas of logic programming and artificial intelligence, and algorithmic patterns. She has developed academic programs for undergraduate level in computer science. Her primary research interests are computer science educational research, students' conceptualization of computer science, as well as in-service teacher education and distance learning.

Tammy Rosenthal received her PhD degree in Science Teaching from the Science Faculty of the Hebrew University. She wrote her PhD thesis at Stanford University and the research was carried out as a joint project of the two universities, with two advisors in each. She is currently a lecturer in the Department of Computer Science in the Holon Institute of Technology, teaching C Programming and Introduction to System Programming. She has also worked in the USA for seven years, developing multimedia online courses in computer science for the Education Program for Gifted Youth at Stanford University, and has continued this development work for Stanford since her return to Israel (C courses at http://www-epgy.stanford.edu). Her primary research interests are the development of automated evaluation methods for computer programs and developing computer science courses for distance learning and e-learning.

John English is a senior lecturer in Computer Science at the University of Brighton. He graduated with an honours degree in Computer Studies in 1978 and has since received a PhD by publication in Computer Science Education. John has worked in higher education for over 30 years, and during this time has published numerous papers in the field of computer science education as well as two textbooks (‘Ada 95: the Craft of Object Oriented Programming’ and ‘Introduction to Operating Systems: Behind the Desktop’). He has also undertaken consultancy work for industry and is the author of several well-known software products for the education sector. His current research interests are centred in the field of automated assessment.