NUI MAYNOOTH

Ollscoil na hÉireann Má Nuad

National University of Ireland, Maynooth
MAYNOOTH, CO. KILDARE, IRELAND.

DEPARTMENT OF COMPUTER SCIENCE,

TECHNICAL REPORT SERIES

Enhancing Skills Transfer through Problem-based Learning

Jackie O'Kelly, Rosemary Monahan, J. Paul Gibson and Stephen Brown

NUIM-CS-TR-2005-13

Enhancing Skills Transfer through Problem-based Learning

Jackie O'Kelly, Rosemary Monahan, J. Paul Gibson and Stephen Brown
Department of Computer Science, NUI, Maynooth
{jokelly, rosemary, pgibson, sbrown} @cs.nuim.ie

ABSTRACT

Problem-based Learning (PBL) has proved itself as a successful teaching and learning environment in the medical field, and has slowly become the preferred teaching and learning method in other disciplines. In this report we look at the learning theories that have influenced PBL and investigate the use of PBL in computer science. We extend the boundaries of PBL and software engineering education with a proposal that fully integrates PBL into a computer science and software engineering degree structure. The objective of this proposal is to produce graduates who can successfully transfer their knowledge and skills into practical situations in new domains.

1. INTRODUCTION

Globalisation of both higher education and the work environment, in addition to the increase in the mobility of students and workers poses an increasing challenge to tertiary level education. These factors add a level of complexity to the gap that exists between academia and industry, and between theory and practice. Determining and developing the skills that students need in addition to their software engineering discipline, to compete successfully in a global market is becoming a fundamental requirement for educationalists. In Ireland, the Expert Group on Future Skills Need (2005) have identified a shortage in the number of software engineers due to the slowdown in the Information Technology (IT) sector and a fall in enrolments on software engineering courses. It notes that the skill gap in this area is likely to widen. It also notes that the sector utilises the skills of electronic and electrical engineers, and in many cases, these engineers act as a substitute for software engineers. In this report we present a problem based software engineering environment that integrates all years of the undergraduate and graduate degree programmes in a real world work group setting. The remainder of the report addresses the theoretical foundations surrounding PBL, a description of what PBL is and its role in computer science in addition to our implementation of PBL. In section 7 we put forward our proposed framework and finally draw together our conclusions.

2. THEORETICAL FOUNDATIONS OF PROBLEM-BASED LEARNING

Problem-based Learning is influenced by a number of educational theories, namely Experiential Learning, Constructivism, Enquiry-based Learning and Discovery Learning. The educational philosophy of John Dewey has been one of the major influences in what was termed the 'newer philosophy'. He believed in the unity of theory and practice and stated that there is an *"intimate and necessary relation between the process of actual experience and education"* (Dewey, 1967).

2.1 Experiential Learning

Kolb (1984) ties the intellectual origins of his work in Experiential Learning to that of John Dewey, Kurt Lewin and Jean Piaget. He sees Experiential Learning as the process that links education, work and personal development (Figure 1). The emphasis on the process of learning as opposed to the behavioural outcomes distinguishes experiential learning from the idealist approaches of traditional education. Ideas are formed and re-formed through experience so that learning becomes a continuous process grounded in experience.

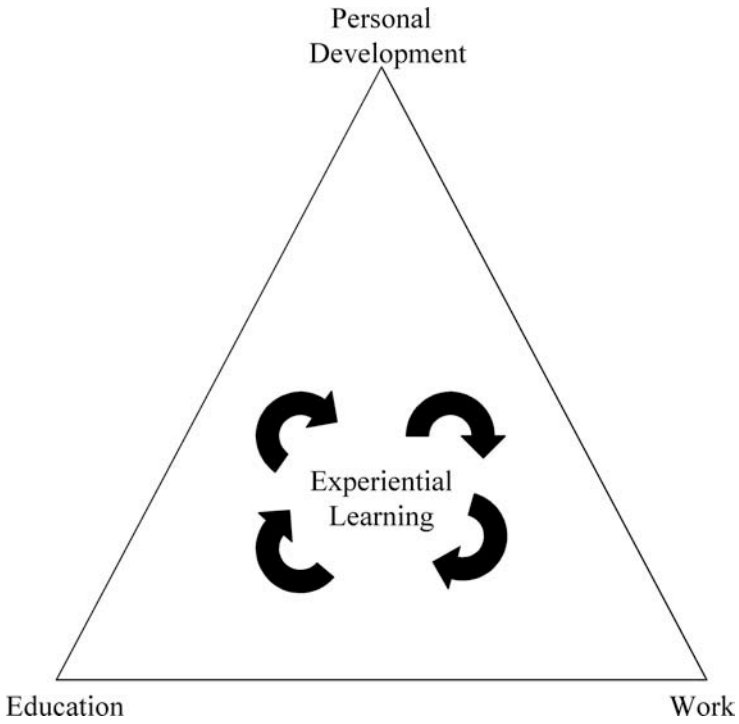

Figure 1: Experiential Learning as the Process that links Education, Work and Personal Development

2.2 Constructivism

Theoriests John Dewey, Lev Vygotsky, Jean Piaget, Jerome Bruner, Seymour Papert and Mitchell Resnick are associated with the theory of constructivism. Constructivists believe that all humans have the ability to construct knowledge in their own minds through a process of discovery and problem-solving. The extent to which this process can take place naturally, without structure and teaching is the

defining factor amongst the advocates of this learning theory (Forrester & Jantzie, 2005).

2.3 Enquiry-based Learning

Enquiry-based Learning is an approach to learning that involves a process of exploring the natural or material world, and that leads to asking questions, making discoveries, and rigorously testing those discoveries in the search for new understanding. The enquiry process is driven by one's own curiosity, wonder, interest, or passion to understand an observation or solve a problem. This theory is also influenced by the work of Dewey, who believed that children should learn from direct experience and that their natural curiosity should be cultivated. Jean Piaget and Jerome Bruner subsequently added the weight of cognitive research to Dewey's philosophical propositions.

2.4 Discovery Learning

Discovery Learning is the learning that comes through free activity in a rich environment, only mildly structured by the teacher to facilitate learning (Hilgard & Bower, 1975). The concept of discovery learning has appeared as a part of the educational philosophy of Dewey. It also enjoys the support of learning theorists and psychologists Piaget, Bruner, and Papert. Bruner was influential in defining Discovery Learning. *"Emphasis on discovery in learning has precisely the effect on the learner of leading him to be a constructionist, to organize what he is encountering in a manner not only designed to discover regularity and relatedness, but also to avoid the kind of information drift that fails to keep account of the uses to which information might have to be put"* (Bruner, 1962). Discovery learning is based on the assumption that education is a process, not a set of facts.

3. TRANSFER OF LEARNING

Transfer of learning occurs when a person's learning in one situation influences their learning and performance in other situations. The question for educationalist is "*in what way and to what extent will acquisition of skills, knowledge, understanding, behaviours and attitudes in one subject or learning situation influence performance or learning in other subjects and situations?"* (Bigge & Shermis, 1999). Educational psychologist Charles Judd recognised two possible levels of learning – rote memorisation with little, if any, meaning (in which he placed little value) and generalised knowledge with many intellectual associations (in which he placed a high premium). He stated: *"when new skills are cultivated by an individual, the muscles are brought into coordinated action through elaborately organised patterns developed in the nervous system"* and *"Generalisations which epitomise great numbers of experiences are the highest products of racial and individual intellectual effort"* (Bigge & Shermis, 1999).

Employers are increasingly looking for workers that not only have multi-disciplinary skills but also have the ability to apply their skills in an industrial environment. The benefits of skills such as communication, team-work, planning, problem solving, critical thinking and negotiation are not realised unless the students are able to transfer these to an industrial setting. This also applies to the basic skills in Computer Science and Software Engineering. The Expert Group on Future Skills Needs (2004)

identified a major problem in graduates qualifying without any fundamental understanding of their discipline, and – as a consequence – an inability to apply what they had learned in University to real-world problems.

4. WHAT IS PROBLEM-BASED LEARNING?

The fundamental principle supporting the concept of PBL is that **the problem** is the driving force that initiates the learning. This way of learning encourages a deeper understanding of the material rather than superficial coverage; which supports the views on learning outlined by Judd in section 3 above. While there is no universal view or definition of what constitutes PBL we present practitioners' definitions of PBL covering three decades. PBL was defined by Barrows & Tamblyn (1980) as "*the learning which results from the process of working towards the understanding of, or resolution of, a problem. The problem is encountered first in the learning process".* Woods (1994) defined it as "a*n approach to learning that uses a problem to drive the learning rather than a lecture with subject matter which is taught."* Torp & Sage (2002) define it as *"Focused, experiential learning (minds-on, hands-on) organised around the investigation and resolution of messy, real-world problems."* While models of PBL vary considerably there are a set of common characteristics:

- The problem acts as the catalyst for learning.
- Learning is student centred.
- Learning occurs in a small group setting.
- The lecturer/tutor acts as a facilitator or guide to the learning process.
- Prior knowledge is activated
- New knowledge is acquired.
- Students take responsibility for their own learning.

PBL was pioneered at Case Western Reserve University in the early 1950s (Boud & Feletti, 1998), and in the 1960s by Howard Barrows at McMaster University. The learning approach adopted by McMaster was used as a model for other PBL programs, and is still used as a benchmark for PBL.

A comprehensive literature review of PBL in the medical discipline was conducted by Albanese & Mitchell (1993) and found that in comparison to conventional instruction, "*PBL is more nurturing and enjoyable; PBL graduates perform as well, and sometimes better, on clinical examinations and faculty evaluations. Faculty tend to enjoy teaching using PBL*." Another study undertaken by Vernon & Blake (1993) also in the field of medicine found that "*PBL was significantly superior with respect to students' program evaluation, faculty attitudes, student mood, class attendance, academic process variables and measures of humanism*." Kaufman (2000) asserts that "*PBL has been one of the most successful innovations in medical education and has established its credibility*."

5. PROBLEM-BASED LEARNING IN COMPUTER SCIENCE

Computer Science as a discipline has been and continues to be regularly challenged, primarily due to the rapid evolution of the field. The dynamics involved with the continuous improvement in technology puts additional pressure on educational

institutions to adopt explicit strategies for responding to change. Ellis et al. (1998) argue that the computing discipline lends itself to Problem-based Learning and matches these characteristics in the following ways:

- Computing, is for the most part problem driven;
- Life-long learning is a necessity due to the rapidly and continually changing nature of the industry;
- Practitioners must constantly update their skills and competencies in order to keep abreast of new technology;
- The project groups is the predominant mode of operation within the industry; and
- Computing crosses discipline boundaries.

The Computing Curriculum (2001), developed by the ACM, states that "*Computer science education, must seek to prepare students for lifelong learning that will enable them to move beyond today's technology to meet the challenges of the future*." Pike & Barber (2003) argue that the core competencies that PBL instills are particularly relevant for computer science graduates. However, a study into the use of Problem-based Learning in the teaching of computing within higher education found that PBL is not widely used (Beaumont et al., 2004). In a number of cases where PBL is used it is in a single module, motivated by the enthusiasm of individual tutors. In only one instance was PBL implemented in half of the curriculum. The module in which PBL is most often applied is the programming module (Greening et al., 1997, Fekete & Greening, 1998, Barg et al., 2000, Duke et al., 2000, Adams et al., 2001, Pollock & Jochen, 2001, Beaumont & Fox, 2003, Stevenson, 2003). Kay & Kummerfeld (1998) combine the learning of programming and HCI while Davis et al. (2004) combine it with computer graphics in a large-scale problem. According to Oriogun & Georgiadou (2000) software engineering education is fundamentally a problem-based activity, and they propose a Problem-based Learning grid to facilitate this learning whereas Uden (2003) proposes a problem-based task knowledge structure method to teach students how to conduct their final year projects. Lambrix & Kamkar (1998) use PBL to integrate computer science as part of engineering education and Hämäläinen (2004) developed a purely theoretical problem-based 'theoretical foundations of computer science' course.

There appears to be no argument that students require fundamental skills in order to function effectively in a PBL environment, as identified by (Greening, 1998, Fekete & Greening, 1998). Preliminary findings by McCracken & Waters (1999) indicate that students tend to rely excessively on existing knowledge and they focus almost solely on product-related issues versus process-related ones and that they need explicit training in group dynamics. Beaumont & Frank (2003) argue that skills, attitudes and behaviours take time to develop and that a single semester-long PBL module is unlikely to make a significant difference. Kinnunen & Malmi (2004) describe the characteristics of efficient and inefficient PBL groups. They found that efficient working groups are those where members participate in group meetings and make themselves responsible, come prepared, maintain an open and relaxed atmosphere and encourage each other. Inefficient working groups are those where: there is little participation or attendance, which in turn reduces motivation; or a number of members are free riders who let others do the work; or other members have very strong opinions and there is no understanding or consensus on how to work.

Both Ellis et al. (1998) and Greening (1998) identify that considerable scaffolding is required in order to make the transition from the traditional pedagogical model to a PBL model.

6. PROBLEM-BASED LEARNING AT NUI, MAYNOOTH

The approach to the implementation of PBL into the computer science curriculum at the National University of Ireland Maynooth (NUIM) has been bottom-up; with individual lecturers interested in PBL introducing it into their own module(s). Currently PBL is our method of instruction in specific modules in first, second and third year of the undergraduate degree as well as specific modules in a post-graduate taught Masters programme. However, its implementation has been conducted in a disjoint fashion with no continuity throughout the degree structure.

6.1 PBL in First Year

The programming module is taught over two 12-week semesters, with 6 contact hours per week. In order to try to overcome the inherent difficulty first year students have with programming the module lecturer introduced a PBL workshop into the module, changed the approach used during lecture time and changed the relative ordering of lecture, lab and workshop. The overall objectives of the changes were to:

- Develop the students' problem solving skills,
- Develop the students' critical thinking skills,
- Encourage alternate approaches to problem solving through group work, and
- Encourage deep learning approaches.

At the beginning of the year every student in the class was assigned to a 'formal group' by the lecturer. The group sizes ranged from five to seven on average. Each group was allocated a space to work in, a facilitator, a PBL journal and scaffolding resources. The PBL journal updated weekly by the groups contained a master record of their collaborative work during the year. Each 'formal group' worked as a team for an entire semester. In the first week of term these students undertook an induction program, specifically designed to introduce them to the concept of teamwork.

The students were presented with a problem in the workshop first. The following restrictions were imposed in the workshops:

- The students were not allowed to use computers, and
- They were not allowed to write a Java program as the solution to the problem.

As a group they were required to develop an algorithm, that is, 'a step-by-step set of instructions to solve the problem'.

The alternate approach to lecture time is informed by the work of Deek & Kimmel (1993), Woods (1996) and Waite et al. (2003). A problem is presented at the beginning of class; the students are placed in 'informal groupings' and asked to generate possible ideas to solve the problem with the lecturer facilitating the group

process. The lecturer then collaborates with the students to solve the problem algorithmically with ideas generated from different groups of students. Once a solution to the problem is drafted, the lecturer then steps through the solution with the students, any difficulties are identified and rectified by the class and the step-through process begins again until such time as a viable solution is reached. At this point the translation of the algorithm to code occurs. During this process any programming concepts that students do not know are flagged. At least one of the lecture time slots each week is given to the theory and concepts behind the learning outcomes of the PBL workshops in addition to the flagged programming concepts identified in the lectures (O'Kelly, 2005).

The scheduled lab time involves the students implementing solutions to programming problems. As the term progresses these problems come directly from the PBL workshop. Demonstrators are available to assist the students during this time. However we noted less reliance on the demonstrators and more interaction between the students themselves in solving these problems in the labs.

6.2 PBL in Second Year

The module is taught over a period of 12 weeks. Every week there are two 1-hour lectures. Also, every week – except the first and last – there is a single 2-hour laboratory practical session. Students receive immediate feedback on their performance in the laboratory (they are given a mark out of 10 before they leave the laboratory).

Every week there is a laboratory PBL session where students work as individuals or in teams (students select their own team members) in order to solve a problem (using a computer programme). In general, the following lecture is used to review what the students were supposed to have learnt from the problem; and this follows a more traditional style. The lecture after that follows the *interactive* model, and attempts to re-use the skills that are acquired in solving the previous problem in preparing for the next problem. The lecturer creates 'informal groups'; with some of the groups solving the problem while others observe the problem solving process. However, this approach typically forces the lecturer to follow a week-long cycle of reviewing the previous week's work in order to develop a new problem that best addresses the needs of the class for the next week: what the students learn, from a particular problem, does not always correspond to what the lecturer thinks they would, or should, learn. Some problems are ineffective: the students appear to learn nothing from them and as there is an interdependency between problems, ineffective problems often have to be re-engineered and presented to the students for a second (or sometimes third) time. Often, 1 hour of standard lecturing is not enough time to cover the material required by the module curriculum and the students are concerned that they do not have a standard set of lecture notes from which to revise the material (O'Kelly & Gibson, 2005).

6.3 PBL in Third Year

The course module is spread over a 12-week period and consists of 4 hours per week contact time with mentors and 4 hours per week independent work. Students are assigned to formal groups at the beginning of the module based on the weak-strong selection technique. The average group size is 4, which allows for a total of 384 man-hours to complete the project. The project is the first large problem the students

encounter and typically tends to include databases, web authoring, software engineering, networking, operating systems and programming. The breadth of the problem means that a complete solution is very difficulty to achieve within the allowed time without adopting a rigorous software engineering approach and good project management. The weekly lecture contact time is concentrated in a single half-day session. This gives the students a relatively long period of time to work together and provides them with an opportunity to meet the clients (tutors) under controlled conditions. Attendance is compulsory for the first hour, after which the students are free to choose a location that best suits their team. At the end of the project four deliverable components are required and assessed: the product itself, the final presentation, individual student journals and feedback forms (conducted every two weeks) to assess individual and teamwork skills (Delaney & Mitchell, 2002, Mitchell & Delaney, 2004).

6.4 PBL in MSc in Computer Science (Software Engineering) Programme

The module called 'Testing and Benchmarking Strategies' takes place over a 2-week period: one week of lectures and workshops followed by one week of assessed practical work. Students form groups of 3 to 4 for the first week, but then do their marked practical work independently. The first week is a mixture of lectures, group-work, workshops, and reading. The reading material is given for the evenings, and provides the background knowledge that the students need for the various problems presented. The lectures are used to provide guidance, for example in new tools, or to review the reading material. In some cases new material is also introduced through lectures. The group work is where the majority of the learning takes place; each student team works out how to apply testing principles and techniques to particular testing problems. In the workshops these solutions are presented by the groups, and then discussed by the class. Differences in their approaches are discussed. Given the more mature nature of the students, a 'hands-off' approach is taken to mentoring the groups, with typically just the lecturer or a single assistant working their way round the groups to make sure that they are working effectively, and perhaps (normally via questions) to redirect or refocus a group's activity.

7. A PROPOSED FRAMEWORK

In this section we describe our proposal for a new framework that will add cohesiveness and continuity to PBL within the degree programme. Our proposed framework pushes the boundaries of PBL to a model where PBL is used in a real world problem that spans a software engineer's education.

Our framework presents a **problem-based software engineering** stream that will span all years of a Software Engineer's education through to Masters level. The student will work in an environment that mirrors real world software development where project teams will be divided into sub-teams, each with a particular task. In addition, members of the team will be of varied skill providing **an apprenticeship model** where students can learn from those that have more experience than those who are new to the task.

We model our framework on an undergraduate degree and a Masters level degree currently offered at NUIM: the BSc Computer Science and Software Engineering and

the MSc (Software Engineering). During the four-year undergraduate degree program a student will participate in many software development projects, normally one per academic year. Students should update their software engineering **portfolio** on the completion of each strand of the framework. The result is a portfolio that details their experience of every phase of the software development cycle. Degree level students will focus on requirements analysis, software design, software implementation, software testing, and software maintenance. Students at Masters level will focus on the project management aspects of the project. This portfolio should prove invaluable when assessing a software engineer's skills and their knowledge of the software development process.

We present, in table 1, four parallel streams that provide a framework for Problem-based Software Engineering.

A Framework for Problem-based Software Engineering			
Problem-based Learning	**Apprenticeship Model**	**Student Portfolio**	**Software Development** (Expanded below in Table 2)

Table 1. Parallel Streams in Problem-based Software Engineering

It is not appropriate, particularly in the early stages of a student's education, to assign a complete phase of the software development to one student cohort. Hence, our framework divides the design, implementation and testing phases into manageable strands that the students will learn through. Table 2 describes what should be achieved at each phase of the project.

Software development phases are allocated to students based on the students experience and their learning requirements. To assist in this allocation the framework associates **prerequisites, learning outcomes** and **available resources** with each project phase:

- Prerequisites refer to the experience/ability that a student should have before commencing a software development phase. These prerequisites ensure that a student will have the required background so that they learn to their full potential on each phase of the project.

- The learning outcomes correspond to the knowledge that the student will achieve through the Problem-based Learning approach to each phase of the project. On achieving these outcomes the student should thoroughly understand each phase of software development.

- Every project phase will have resources available to assist the student in their Problem-based Learning. Such resources are normally allocated by the project

manager and will include courses, tutorials, software tools and techniques. As this software engineering stream will be available as part of a wider education program some of these available resources will be courses that are run in parallel with a student participating in a software development phase.

Software Development Phases	Outputs
Project Management	Management Report Project Milestone Reports & Presentations.
Requirements Analysis	Requirement Analysis Document
Software Design	**Design Strand 1: High Level Design** System Design Documents e.g. Architectural, Subsystems and GUI designs **Design Strand 2: Low Level Design** Detailed Design Documents e.g. Component, Class, Method designs
Software Implementation	**Coding Strand 1: Basic Programming** Code (Basic method implementations) with corresponding documentation **Coding Strand 2: Advanced Programming** Software Product (Complete Code with corresponding documentation)
Software Testing	**Testing Strand 1: Test Case Generation** Test Suite Documents **Testing Strand 2: Test Log** Updated Test Log
Software Maintenance	Revised System with Version Control Documentation and Updates

Table 2. Software Development Phase Outputs

The framework details are presented below, in tables 3 to 7, with suggested allocations of software development phases to suitable student cohorts.

7.1 Problem Based Framework for a Software Engineering Programme.

7.1.1 Undergraduate Programme:

Students entering an undergraduate degree program will have little or no knowledge of the software development process. During their initial year of the course they will learn about basic programming skills, the importance of a structured approach to problem solving as well as the importance of software design and testing documents. Available resources include courses on programming principles, design and test document comprehension, as well as the project manager who will have knowledge about the overall project in which the student is involved. Table 3 describes the Problem-based Learning stream for software engineering in year 1 of an undergraduate degree program:

Student Cohort	Software Phase	Phase Prerequisites	Learning Outcomes	Resources Available
Computer Science / Software Engineering Undergraduate Level: Year 1/ Freshman Year	Coding Strand 1: Basic Programming	Ability to take instruction	Basic Programming Skills Language Syntax Correctness Documentation Structured approach Reading and understanding design documentation	Design Document Comprehension Programming Principles Project Manager
	Testing Strand 2: Test Log	Ability to take instruction	Reading and understanding test documentation How to update test logs	Test Document Comprehension Project Manager

Table 3. Software Engineering Stream: Undergraduate Year 1

On progression into the second year of the program the student will learn more in-depth programming skills, focusing on the correct and efficient implementation of software designs. As part of the apprenticeship model, the student will allocate some of the basic programming tasks, e.g. simple method implementations, to students who are allocated Coding Strand 1. These students will teach basic programming skills to the more junior programmers, while learning through their implementation of design documents provided by more senior undergraduate students. The advanced programming strand and its allocation is presented in Table 4 as follows:

Student Cohort	Software Phase	Phase Prerequisites	Learning Outcomes	Resources Available
Computer Science / Software Engineering Undergraduate Level: Year 2/ Sophomore Year	Coding Strand 2: Advanced Programming	Reading and understanding design documents. Strong grounding in Programming Principles.	Implementation of design documents. Coding Skills (Correctness, Syntax, Efficiency, Documentation Structured approach, Ability to break a task into relevant subtasks) Algorithms and Data structures	Design Document Comprehension Programming Principles ||[1] Communication Skills || Software Testing Project Manager

Table 4. Software Engineering Stream: Undergraduate Year 2

Having learned how to interpret and implement design documents, students should learn how to generate their own design documents. The first step towards this process is generating a requirements analysis document. As communication skills are a vital component of this process a course on communication skills should be provided in year 2 (indicated by || in the available resources listing). Resources available should include instruction on techniques such as use cases, context diagrams, requirement traceability matrices and screen prototypes; while learning outcomes should include coverage, completeness, conciseness and clarity of the system requirements as well as both critical and analytical thinking.

The second step is the generation of system test cases. These test cases will be used to test the software implementation with respect to its design. Issues such as benchmarking, software complexity, software metrics, system portability and efficiency should be included in the test cases. A course in software testing should be presented to the students in year two to prepare them for the test case generation phase.

Both requirements analysis and test case generation phases are presented below in Table 5:

[1] Indicates a module on this topic that the student takes in parallel to the project phase / strand that they are currently working on.

Student Cohort	Software Phase	Phase Prerequisites	Learning Outcomes	Resources Available
Computer Science / Software Engineering Undergraduate Level: Year 3 / Junior Year	Requirements Analysis	Communication Skills Writing / Reporting skills	Analysis of what the software should do. Requirements Analysis Document Generation	Project Manager Requirements Analysis Tools and Techniques Collaborative requirements analysis sessions || Design Concepts
	Testing Strand 1: Test Case Generation	Read and understand Requirements Analysis and the design documents	What, Why, How of testing Identification and creation of test cases Documentation and reporting test cases and test log mechanisms.	Project Manager Testing Strategies and Tools

Table 5. Software Engineering Stream: Undergraduate Year 3

When students reach their final year they should be extremely familiar with design documents and how to implement them. They should be ready to learn how to generate a complete, concise and clear design document. Under our framework all design documentation is allocated to fourth year undergraduate students as follows, in table 6.

We note that due to a dependency between phases that some phases will be scheduled earlier in the year than others; requirements analysis and design documentation will require completion during the early part of the year while system implementation and testing will be completed during the later part of the year.

Student Cohort	Software Phase	Phase Prerequisites	Learning Outcomes	Resources Available
Computer Science / Software Engineering Undergraduate Level: Year 4/ Senior Year	Design Strand 1: High Level Design	Have completed a requirements analysis document.	How to make design decisions at architectural, system and interface levels.	Software Engineering Tools and Techniques Project Manager
	Design Strand 2: Low Level Design	Have completed a requirements analysis document	How to make design decisions at coding levels (software components, classes, methods...)	Software Engineering Tools and Techniques Project Manager

Table 6. Software Engineering Stream: Undergraduate Year 4

7.1.2 Postgraduate Programme:

Postgraduate Masters students have a primary degree in computer science, software engineering or a related discipline. Hence, it is most likely that they will have experience of many phases of software development but no experience of project management. The skills that these students will learn through their project management phase include the management of the project, people, time, finances and other resources. Other skills such as reporting, communication, presentation, delegation, negotiation, motivation, facilitation, and performance review will also be essential learning outcomes. The MSc in Computer Science (Software Engineering) at NUIM is one year in duration where the students will undertake the management of a large software project that is designed, implemented and tested by undergraduate students from the computer science and software engineering degree program. Details are as follows, in table 7:

Student Cohort	Software Phase	Phase Prerequisites	Learning Outcomes	Resources Available
Computer Science / Software Engineering Graduate Level: MSc	Project Management	Experience of at least one phase and an understanding of all phases of the software development process	Management skills Identify software development Phase Boundaries and phase overlaps	Course Facilitator Experts in relevant domains (Academic Staff, Industrial Partners, Clients ...)

Table 7. Software Engineering Stream: Graduate MSc

As the above framework illustrates, each project team will span all years of an undergraduate degree programme as well as a Masters level programme. This team structure will strengthen the apprenticeship model where students will learn from their peers. This model is further strengthened through mixing students of different abilities on the requirements and design, implementation and testing teams. It is hoped to extend this apprenticeship model so that students are rewarded by progressing to more interesting problems. This will help to maintain student motivation and enthusiasm as well as mirroring the real world industrial experience.

The only software development phase that has not been allocated to a student cohort within this framework is software maintenance. Software maintenance tasks will be allocated by the project manager depending on their impact on the overall design, implementation and testing phases of the system software. Each team, which is allocated a task associated with system maintenance, will be expected to generate a revised system with version control documentation and updates.

As a student may participate in five years worth of project work, their completed portfolio may span five large software projects and all phases of software development.

8. Conclusions

In this report we have identified a skills shortage in the software engineering industry in addition to a deficit in the skills required by software engineering students to compete in a global economy. We presented Problem-based Learning as a candidate approach to address these problems and provide the theoretical foundation to support this approach. We extend the boundaries of PBL and software engineering with our proposed framework that provides a challenging real-world environment where students can prosper. Our proposal encourages active and lifelong learning and provides a real world experience through the use of work groups that integrates students from all years of an undergraduate and graduate software engineering degree

programme. The work group environment, project management, the apprenticeship model and the student portfolio will develop and enhance the technical and 'soft' skills that are required for software engineers to work effectively today.

REFERENCES

Adams, M., Clarke S., and Thomas R., (2001), *Developing Graduate Capabilities Through PBL*, Third Asia Pacific Conference on Problem Based Learning, December 2001, Rockhampton, Queensland.

Albanese, M.A., and Mitchell, S., (1993) *Problem-based Learning: A Review of Literature on Its Outcome and Implementation Issues*. Academic Medicine, Volume 68, Number 1, January.

Barg, M., Fekete, A., Greening, T., Hollands, O., Kay, J., Kingston, J.H. and Crawford, K., (2000), *Problem-Based Learning for Foundation Computer Science Courses*, Computer Science Education, Vol. 10, No. 2, pp. 109-128.

Barrows H.S., and Tamblyn R.M. (1980) *Problem-Based Learning: An Approach to Medical Education*. Springer Publishing Company, New York, ISBN 0826128408 p.1.

Beaumont, C., and Fox, C., (2003), *Learning Programming: Enhancing Quality Through Problem-Based Learning.* 4th Annual LSTN-ICS Conference, NUI, Galway.

Beaumont, C., and Frank, B., (2003), *Enhancing Employability through Problem-based Learning*, Delivering Employability Conference, UCLAN 9th April, 2003.

Beaumont, C., Sackville, A., and Cheng, C.S., (2004), *Identifying Good Practice in the use of PBL to teach computing*, ITALICS 3 (1), LTSN-ICS.

Bigge, M., and Shermis, S., (1999) *Learning Theories for Teachers*, Sixth Edition, Addison Wesley Longman, Inc., ISBN 0-321-02343-9.

Boud, D.,and Feletti, G., (1998) *The Challenge of Problem-Based Learning*, 2nd Edition, Kogan Page, London, ISBN 0-74942-560-1.

Bruner, J.S. (1962). *On knowing: Essays for the left hand*. Harvard University Press, Cambridge, Mass, ISBN 0-6746-3525-6.

Computing Curricula 2001, Computer Science volume, Final Report, December 15, 2001, The Joint Task Force on Computing Curricula IEEE Computer Society, Association for Computing Machinery.

Davis, T., Geist, R., Matzko, S., and Westall, J., (2004) τεχνη*: A First Step*, SIGCSE 2004, Norfolk, Virginia, USA.

Deek, F.P., and Kimmel, H., (1993), *Changing the Students' Role from Passive Listeners to Active Participants.* Frontiers in Education Conference, pp. 321-325.

Delaney, D., and Mitchell, G., (2002), *PBL Applied to Software Engineering Group Projects*, ICTE 2002, International Conference on Information and Communication Technologies in Education.

Dewey, J., (1967), *Experience and Education*, Kappa Delta Pi, 1938, Collier Books, New York.

Duke, R., Salzman, E., Burmeister, J., Poon, J. and Murray, L., (2000), *Teaching Programming to Beginners – choosing the language is just the first step.* ACE 2000, 12/00, Melbourne, Australia.

Ellis, A., Carswell, L., Bernat, A., Deveaux, D., Frison, P., Meisalo, V., Meyer J., Nulden, U., Rugelj, J., and Tarhio, J., (1998), *Resources, Tools, and Techniques for Problem Based Learning in Computing*, Report of the ITiCSE'98 Working Group on Problem Based Learning.

Expert Group on Future Skills Needs, (2004) Forfás submission to the Your Education System Review. Available online at http://www.forfas.ie/publications/_list/skills.html

Expert Group on Future Skills Needs, (2005) National Skills Bulletin 2005. Available online at http://www.skillsireland.ie/press/reports/index.html

Fekete, A. and Greening T., (1998), *Conveying Technical Content in a Curriculum Using Problem Based Learning*, ACSE'98, Brisbane, QLD, Australia.

Forrester, D., Jantzie, N., *Learning Theories*, accessed on-line August 17th, 2005 at http://www.acs.ucalgary.ca/gary.ca/~gnjantz /learning_theories.htm

Greening, T., Kay, J., Kingston, J., (1997) *Results of a PBL Trail in First-Year Computer Science*. ACSE'97, Melbourne, Australia.

Greening, T. (1998). *Scaffolding for Success in Problem-Based Learning*. Med Educ Online [serial online] 1998;3,4. Available from http://www.Med-Ed-Online.org

Hämäläinen, W., (2004), *Statistical analysis of problem-based learning in theory of computation*, Proceedings of the Fourth Finnish/Baltic Sea Conference on Computer Science Education, October 1-3, 2004, Koli, Finland.

Hilgard, E., Bower G., (1975) *Theories of Learning*, Fourth Edition, Prentice-Hall Inc., Englewood Cliffs, New Jersey, ISBN 0-13-914457-9.

Kaufman D., (2000) *Problem-based learning - time to step back?* Medical Education, Vol 34, Issue 7: 509-511.

Kay, J., and Kummerfeld, B., (1998), *A Problem-based Interface Design and Programming Course* (1998), SIGCSE 98, Atlanta, GA, USA.

Kinnunen, P., and Malmi, L., (2004), *Do Students Work Efficiently in a Group? – Problem-Based Learning Groups in Basic Programming Course*, Proceedings of the Fourth Finnish/Baltic Sea Conference on Computer Science Education, October 1-3, 2004, Koli, Finland.

Kolb, D., (1984) *Experiential Learning*, Prentice-Hall Inc, Englewood Cliffs, New Jersey, ISBN 0-13-295261-0.

Lambrix, P., and Kamkar, M., (1998), *Computer Science as an Integrated Part of Engineering Education*, ITiCSE '98, Dublin, Ireland.

McCracken, M., and Waters, R., (1999), *WHY? When an Otherwise Successful Intervention Fails*, ITiCSE '99 6/99 Cracow, Poland.

Mitchell, G., and Delaney, D., (2004), *An Assessment Strategy to Determine Learning Outcomes in a Software Engineering Problem-based Learning Course*, International Journal Engineering Education, Vol. 20, No. 3, pp. 494-502, Tempus Publications, Great Britain.

O'Kelly, J., (2005), *Designing a Hybrid PBL Course: A Case Study of First Year Computer Science in NUI, Maynooth* in Handbook of Enquiry and Problem-based Learning: Irish case Studies and International Perspectives, pp. 43–53. CELT, NUI Galway, ISBN-13 978-0-9551698-0-9

O'Kelly, J., and Gibson, J.P., (2005) *PBL: Year One Analysis—Interpretation and Validation*, PBL International Conference 2005, PBL In Context – Bridging work and Education, Lahti, Finland.

Oriogun, P.K., and Georgiadou, E., (2000), *Towards Ensuring the Development of Capabilities Through the Use of the Problem Based Learning Grid*. 8th Annual Conference on the Teaching of Computing, Edinburgh.

Pike, A., and Barber, D., (2003) *A Preliminary Investigation of the Role of Problem Based Learning (PBL)*, ITB Journal, Issue Number 8, December 2003.

Pollock, L., and Jochen, M., (2001), *Making Parallel Programming Accessible to Inexperienced Programmers through Cooperative Learning*. SIGCSE 2001, 2/01, Charlotte, NC, USA.

Stevenson, S., *Problem Solving Principles and Free-Writing Techniques in a Computer Science Class* (2003) ACMSE'03, 41st ACM Southeast Regional Conference, Savannah, Georgia, March 2003.

Torp, L., and Sage, S., (2002) *Problems as Possibilities: Problem-Based Learning for K–16 Education,* 2nd Edition, Association for Supervision and Curriculum Development (ASCD), Alexandria, VA, USA, ISBN 0-87120-574-2.

Uden, L., (2003), *Problem-Based Task Knowledge Structures in Projects*, 4th Annual LTSN-ICS Conference, NUI, Galway.

Vernon D., and Blake R., (1993) *Does Problem-based Learning Work? A Meta-Analysis of Evaluative Research.* Academic Medicine, Volume 68, Number 7, July.

Waite, M.W., Jackson M.H., and Diwan, A.,(2003), *The Conversational Classroom*, SIGCSE 2003, February 19-23, Reno, Nevada, USA.

Woods, D. R., (1996) *Problem-based Learning: how to gain the most from PBL*. Waterdown, Ontario, ISBN 0-9698725-0-X.