Developing Case Studies to Enhance Student Learning

Sally Clarke[1] **Richard Thomas**[2] **Michael Adams**[2]

[1]Faculty Education Unit
Faculty of Medical and Health Sciences
The University of Auckland
Private Bag 92019
Auckland, New Zealand
s.clarke@auckland.ac.nz

[2]Faculty of Information Technology
Queensland University of Technology
GPO Box 2434
Brisbane, Qld. Australia 4001
{r.thomas, m3.adams}@qut.edu.au

Abstract

In this paper we describe the cases developed in response to the multifaceted challenges we faced in educating IT professionals in intermediate-level programming. The challenges included: the practical nature of the subject, the level of skill attained, preparing students for the workplace and class size. We implemented Problem-Based Learning (PBL) as the teaching and learning method to meet these challenges by helping first-year university students learn programming while simultaneously developing graduate capabilities. This necessitated redevelopment of the curriculum and the way it was taught. We combined software engineering and information systems approaches in real-world scenarios. The cases were designed to help students come to terms with the human side of information technology through the context and method of teaching and learning as they worked through the cases.

Keywords: intermediate-level Java programming, problem-based learning, case-study, Swing, unit testing, system design

1 Introduction

The Faculty of Information Technology at QUT believed that it required a new approach to teaching Java programming. Nationally the progression rate in computer science was 76.9% (DEST 2002). Attrition in first-year intermediate-level programming was still similar to national levels despite many small changes to the curriculum and method of teaching implemented. The Faculty considered the content appropriate but considered a new approach to teaching and learning necessary.

Two important pieces of anecdotal evidence impacting on the way IT is taught added to the argument for a new teaching approach. Firstly, many first years are not inclined to seek their tutor's help in practical classes. Secondly, unless tutors learnt about teaching and learning in higher education, when asked a question by a student they often take up the challenge by providing the answer without stimulating the student to think more deeply about the topic.

In redeveloping the first-year curriculum, we acknowledged various issues: the applied nature of the compulsory unit, the current method of teaching, the need to prepare students for the workplace, the level of knowledge and skill attained, and the large student cohort – at the city campus there were typically 350+ students in first semester and 580+ students in second semester.

One of the main goals of the unit was to give students an opportunity to improve their programming skills in a modern language, in this case Java. The unit reinforced the key skills of graphical user interface programming, testing skills, system analysis and design, file handling and database interaction.

Previously the unit was taught as a 50-minute lecture followed later in the week by a 2-hour laboratory or practical class. In this lecture and 'drill' approach students gained exposure to concepts in lectures and were expected to apply them in laboratory classes where they wrote and compiled small programs. This type of learning is at a low level on any taxonomy of learning (Bloom 1964, Bowden and Marton 1999). Our goal was for students to achieve deeper learning outcomes such as comprehension, changed views and behaviour, and avoid encouraging students to adopt a 'chug and plug' approach (Bowden and Marton 1999) to programming.

With an emphasis in higher education on graduate capabilities (in our University, knowledge/problem-solving, ethical/attitudinal and social/relational attributes) (Neilsen 2000), we recognized that in addition to disciplinary knowledge, students needed exposure to developing skills in life-long learning, time management and, more importantly, in interpersonal and communication skills required for interacting with both clients and colleagues.

1.1 Problem Based Learning

We considered Problem Based Learning (PBL) as a suitable teaching methodology since it provided features that addressed many of the challenges we faced. Firstly,

data from medical education (Newble and Clarke 1986) indicated that we could achieve the deeper learning outcomes we were looking for in first year by using the PBL approach. Secondly, the messy, 'real-world' scenarios or 'PBL problems', developed as an essential part of the PBL curriculum, could provide students with a simulated workplace experience and motive to think more deeply about disciplinary knowledge. With our very large class sizes, this simulation was our best approximation to industry experience or internship. Thirdly, in the workplace, much of a programmer's work involves dealing with colleagues and clients. Although the small group size of the PBL tutorials may assist students to develop interpersonal skills so desired by employers (Bentley, et al. 1999, Turner and Lowry 1999), we implemented specific teaching strategies to ensure that students could realise this learning opportunity.

1.1.1 The PBL Process

The PBL process structured the tutorials and guided students through the PBL problem in a way that engaged both students and tutors. We modified the PBL process described by Barrows (1989) to suit our IT context. Our PBL process in the case studies reported here:

- included an opportunity for students to declare their feelings about the professional scenario,
- emphasised student interactions in the first stage of the tutorial,
- did not include explanations of causes of the problem (diagnoses) but proposed and discussed plausible alternative solutions to the problem.

1.1.2 Development of the Curriculum and Cases

From the first author's experience with PBL elsewhere, we agreed that reduction of content knowledge in the curriculum was a key factor for achieving the project goals (Clarke, Thomas & Adams 2001). Hence, reducing the content knowledge became a primary guiding principle in developing the curriculum to enable students to focus on key objectives (Clark 2001, Ramsden 1992). We used guidelines prepared by Searle and Clarke (2000) for developing and writing the cases.

1.1.3 Teaching Teamwork and Communication Skills

We used the 4SAT Step 2B (Zimitat and Alexander 1999) for teaching teamwork and communication skills. This validated instrument provided a comprehensive checklist for assessing interpersonal and communication skills of a group as either satisfactory or unsatisfactory. Where the self-rating was unsatisfactory, group members suggested strategies to improve the group interaction and hence future ratings. We used the instrument formatively as a tutor-guided, student self-assessment instrument at the end of each tutorial in preparation for the two summative ratings performed by tutors in weeks 4 and 7 of the 13-week semester.

2 Case Descriptions

2.1 Case One

The goal of the first case was for students to learn graphical user interface (GUI) programming in Java using the Swing library by taking over an existing, poorly developed prototype and redeveloping it for delivery to a client. We wanted students to have the opportunity to develop a useful product by reusing existing code and start to develop a client focus.

A PBL "trigger" introduced students to the case by indicating that a colleague was ill and that they must complete her work. Students had access to the source code and design documentation for the existing prototype via the unit web site, and examined the existing prototype in the tutorial following discussion of the trigger.

In the first stage of this case, students aimed to learn techniques for evaluating the suitability of a user interface. Upon first viewing the prototype most students commented on the poor quality of the interface, but when pressed to justify their comments they often could not provide a supporting argument other than "it looks bad".

After viewing the prototype students discussed ways to evaluate and determine if it was suitable, or if not, identify any weaknesses. A tutor guided this discussion probing with questions designed to direct students to consider testing the screen with a range of users. The prototype screen embedded in a testing environment facilitated the students' testing of the screen. The testing environment captured the time spent entering data, the number of corrections and the number of invalid entries. The testing environment also collated data from many trial runs with different users. Students conducted this testing out of class and reported on their collated results to the tutor a week later. At this stage of the first case students should have learnt techniques to objectively judge the suitability of a user interface and identify the parts of the prototype screen that cause the most problems with users. Students then had the starting point for the next stage of this case where they need to redesign the interface.

On returning to their tutorials, students reported the results of their testing including presenting the data obtained from the testing environment as well as their interpretation of the data. Following their report students received a new trigger from their tutor.

The form of this trigger was notes from the client – a role played by the tutor. The trigger, designed to confirm the students' conclusions about the faults with the prototype, set the stage for students to consider how to redesign the user interface. At this point in the case students were given a memo from their manager, another role played by the tutor, saying he intended to send them on a Swing training course but places were full until next month.

This trigger, along with guided questioning from the tutor, was designed to guide students to choose a strategy to learn the scope of the Swing library and how to start programming with the components. The desired strategy was for students to work individually or in pairs to

discover the details of a small number of components in the library. Students were left to investigate the library and to consider how the available components could improve the user interface.

At the start of their tutorials in the third week of this case, students were in a position to confidently redesign the user interface. They return to the tutorial to explain to the other team members how to make use of the components they have investigated over the past week. During their explanations they provide a walk-through of sample code demonstrating how to use the components. The team then prepared a design for the improved user interface and plans on how they will implement it over the remainder of the week.

In the following week students presented their improved user interfaces to the other teams in their tutorial session. Each team had five minutes to demonstrate their user interface and explain why their improvements met the design goals arising from their testing and the client's requirements.

2.1.1 Reflections on Case One

This case was very successful, hence, the very few changes to it over the five implementations. This case worked the best because we spent more time in planning it, purposely made it simpler, and gave students longer to work through it as it was the problem used to introduce students to PBL. We encountered difficulties with the scope of one of the triggers and students not thinking more broadly about the work context, as well as with a lack of content for high achieving students.

As first written, the simulated client was a person from a non-English speaking background (NESB). The trigger information provided to students from the client in the second week reflected this, but was too subtle for some students. The trigger, written in "broken-English" to match the NESB profile and to give students the experience of having to decipher such a message, significantly distracted students from the goal. Students took too much time to work through the issues related to dealing with a NESB person. Subsequently we rewrote the information from the client to be clearer and easier to comprehend.

A more significant change to the case in the final two semesters was a bonus trigger introduced at the end of the case asking teams to implement more screens within the overall system's GUI. This trigger was designed and introduced to challenge students who had pre-existing knowledge in the subject area and/or who completed the tasks early. Only teams that had substantially completed the project by the third week received it (less than 10% of groups). A small number of students from groups who were not given the bonus trigger thought that they were disadvantaged. They needed to be reassured that they could achieve full marks on the summative assessment without the bonus trigger and that in all likelihood if they had attempted the bonus trigger they would have spent less time on the core learning objectives and thus achieved a lower mark overall. Most of the groups receiving the bonus trigger appreciated the challenge and took pride in showing off their extra work to other teams.

2.2 Case Two

The goal of the second case was for students to find and fix errors in existing code. In the workplace scenario of this case, a colleague suddenly left the company, indicating that the ecommerce prototype they worked on was almost complete but "has a few bugs". The team had to determine the number of existing errors and assess how serious they were, within a two week deadline.

As in the first case, students were given a trigger introducing the scenario and major goals. The key learning objective for this stage of the problem gave students the opportunity to discover specific techniques to perform white-box unit testing and to apply these techniques to the source code. The problem design required students to report back to the team about the unit testing techniques they learnt and also to describe the errors they discovered in the classes they tested. After this report back, the triggers streamed students depending on their progress. Teams who failed to demonstrate adequate understanding of unit testing received one trigger, while teams who successfully demonstrated understanding of unit testing received a different trigger.

The teams who did not demonstrate an adequate understanding of unit testing received a trigger providing more direction for testing and asked students to complete further testing. Particular emphasis was placed on code coverage during testing, regression testing and test result readability. These issues provided the tutor with discussion points to help the group to understand the benefits of effective unit testing.

The alternative trigger provided teams who met all the key learning objectives in the first week of this case with an opportunity to fix the errors they discovered. This allowed students to see the benefits of unit testing during regression testing by putting them in the situation where they needed to rerun the same tests on the modified code.

Students from both streams produced a master list of errors that they discovered in the prototype at the end of this case. Those teams that were given the alternative trigger also produce a partially fixed prototype. This case emphasised learning how to perform unit testing and some formal testing strategies. Groups that did not debug or attempt to fix errors were not disadvantaged as this was considered beyond the objective of the case.

2.2.1 Reflections on Case Two

This case was not as successful as the first case as almost 20% of students failed to demonstrate the level of competence in unit testing that we desired. In summative assessment on the key learning objectives of this case over 50% of students achieved results higher than 80%, but almost 20% of students achieved results less than 50%. Students were less satisfied with this case. Their feedback indicated they felt the workload was too heavy for two weeks. They also thought it was significantly more difficult than the first case.

This case was implemented five times and underwent several changes in order to improve the outcomes. The program students tested was replaced three times. When a

program was reused, a different set of errors were introduced into the program. The first version of this case did not explicitly direct students to perform unit testing. Tutors questioned students to facilitate them coming to this realisation, based on the requirement to find faults in a program that did not execute and also on a logical use of team members to share the workload. In the tutorials, most teams acknowledged that unit testing was appropriate, but after the tutorial many of these teams instead tried to fix the program to allow each team member to perform individual system testing. They rationalised their actions in the subsequent tutorial saying that unit testing was too time consuming and that they would be able to identify errors more quickly by trying to get a program running. This indicated a misconception about unit testing and roles of testing strategies. Part of this misconception could probably be attributed to the pre-requisite unit and its heavy emphasis solely on system testing. Part of the misconception may have related to how students were led to discover the requirement for unit testing.

In subsequent uses of this case, the tutor took the role of manager and directed the team to perform unit testing. Later iterations of this case further narrowed the direction to perform "white-box" unit testing. In the first implementation of this case there were no lectures, but the second time, a single lecture on testing and quality assurance procedures at an external company were described. The third time, two one-hour lectures provided the concepts of testing and unit testing strategies. The fourth and fifth times the case was used more emphasis was placed on unit testing strategies in the lectures.

We identified a serious problem with students tending to place greater emphasis on fixing errors than on testing. The last change made to the case was to de-emphasise errors correction. Initially the progression of the case was planned for students to perform unit testing and discover errors in the first week and then fix some of the errors in the second week. At the start of their tutorial in the second week students reported on the testing techniques they used and the errors they had discovered. Following this report, originally students were given a trigger to fix the errors. This was changed in the final iteration so that the tutor judged the extent to which students met their learning objectives on unit testing. Those students who had not adequately applied unit testing were given a trigger that more forcefully directed them to perform unit testing. These students were not expected to fix any errors they found. Groups that demonstrated they had met their learning objectives on unit testing were given the original trigger to fix the errors they discovered.

As judged by summative assessment, this approach appeared to improve the case. In the last semester, over 65% of students achieved results higher than 80%, and less than 8% of students achieved results less than 50% for the key learning objectives of this case.

Informal feedback from both students and tutors also indicated that students considered that they understood unit testing better than students in previous semesters. Many students started to apply unit testing in their third case also indicating a significant qualitative improvement.

2.3 Case Three

The goal of the third case was for students to use a structured software development methodology (analysis, design, coding and testing), combined with a project management component. It also asked students to consider ethical issues regarding user access to sensitive data.

Students were required to develop a relatively complex information system to record lodgements of a simplified taxation return and to calculate refunds and liabilities, from an incomplete set of functional requirements. The case incorporated and extended the learning objectives of the previous two cases and introduced students to topics of software design and development, file handling and database connectivity. Unlike the first two cases, where teams were provided with source code to modify and augment, this case required the solution to be designed, coded and implemented from "scratch"..

Five triggers were released over six weeks, with the first three given in the first week. The first provided students with a sizeable and somewhat messy set of functional requirements from which they were to develop a system specification. The second simulated a media release describing the importance that the scenario's client placed on data security issues, and was designed to have students think of ways to incorporate security of data into the specification. The third gave a two week deadline for development of design documentation for the proposed system, key intentions being to provide practice in design and planning techniques, consider how the proposed system met its objectives, equitably assign the various sub-tasks to each team member, and prevent teams from commencing the coding phase before they had a clear design from which to work.

The 'workplace' role of the tutor in this case was a client representative. Once the students' design met with 'client' approval, the fourth trigger was released. This began the coding phase, seeking a solution prototype for demonstration within a two week deadline. This trigger was designed to keep students steadily on task and to help spread the workload evenly over the remaining weeks of the case. Once the students demonstrated and received feedback on a working prototype, the final trigger was released. This trigger requested that the system be completed by the due date.

2.3.1 Reflections on Case Three

The third case was the end result of an amalgamation of three separate cases introduced since the PBL approach was first implemented. Each iteration of case redevelopment followed end of semester reviews.

It was important throughout that the course content of the second half of semester retained its Information Systems (IS) focus. Initially, each major topic area was presented as a separate case, which meant there was a total of five cases presented during semester, three of which were IS cases. Originally, the first of the IS cases reinforced good user interface design principles by asking students to review the GUI prototype developed in Case One and suggest ways to extend it into a simple but functional

working application. The objective was to develop a functional specification and design documentation, thus providing an informally structured introduction to the software development life cycle. The second case was designed to cover software exceptions from both the programmer (explicit handling techniques) and user (appropriate error messaging and interfacing) perspectives. Finally, the third IS case was designed to connect the earlier-developed GUI to a database, and included the benefits and difficulties of database access and the extensibility of such an approach with a view to a web-based system.

Following an end of semester review, four problem areas were identified: lack of lectures, the workload was considered excessive, some groups without prior experience had difficulty developing a functional specification, and students did not enjoy extending the same scenario from Case One through all the IS cases. As a result, the IS cases were reduced from three to two, and a new scenario for each was introduced.

The second version of the first IS case required students to develop a simple word processing system for children. This case gathered the topics of GUI design, file and exception handling and consideration of end user needs. This time, students were provided with a functional specification for the system. After assimilating the specification, students were asked to develop a design and project plan for appraisal and further advice before any coding work could begin. Students were required to verbally defend major design defects, and/or poor plans by providing a prepared rationale. After development of a prototype based on their design, students were asked to draft a set of user testing scenarios for a user test group. The most important aspect of this activity was that students considered the user's perspective. The second IS case had students follow a similar process to develop a database application. Formal lectures were also re-introduced during this period, which, rather than taking a purely didactic approach, combined general advisories on broad concepts and research methods with a-posteriori 'milestones' that reinforced students' perceived direction.

Following further review, all the topic areas covered in the IS half of the semester were amalgamated into a single case which ran for the entire second half of the semester. This case was successfully employed in each of the last three semesters.

3 Outcomes

3.1 Pass Rate

The average semester failure rate for the unit before the introduction of PBL (2001/1) was 26.2%. This is not unexpectedly a little higher than the national data for entire degree programs (DEST 2002) as first year has the highest attrition rate of the three-year degree course. Since PBL was introduced, the average semester failure rate has been 7.8%, which is well below both the unit's previous mean rate and the faculty's first year mean rate. Figure 1 below shows the reduction in failure rate from before the introduction of PBL (The y-axis represents tens of percent failing the subject and the x-axis represents semesters from 1998 to 2002). When analyzing the data a large outlier is obviously noticeable for semester one 1999. That semester marked a change in teaching language (from C to Java) which may have attributed to the 'spike'. But even ignoring this data point, the average semester failure rate for the unit was over 20% prior to the introduction of PBL. Even if the two data points from 2000 are considered to be a downward trend for the failure rate (and statistically they do not show a significant downward trend), the dramatic reduction in the failure rate in the following semesters is still significant.

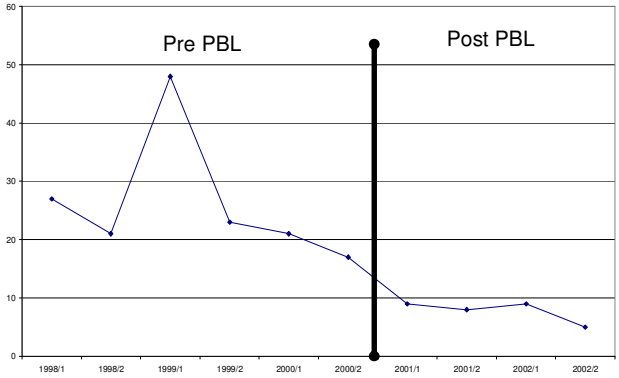

Figure 1: Failure Rates (%) for each Semester 1998/1 to 2002/2

Across the three semesters in which percentage scores have been awarded for the assessment of the PBL cases (2001/2 to 2002/2), over 75% of students achieved results higher than 80%, and over 95% of students achieved results higher than 50%. Summative assessment scores for the IS cases show that just under 50% of students received better than 80% and 86% of students achieved results of 50% or greater, ,which indicates the relative density of the IS case compared to those earlier in the semester.

These figures indicate that the PBL approach, which appealed to the diversity of students, allowed a far greater proportion to achieve a good understanding of the topics covered, and consequently better grades overall. In further support of this claim, informal feedback from follow-on units indicates that students are better prepared for these units than they were prior to the introduction of PBL in this unit. This feedback has been from lecturers and tutors who have worked closely with groups of students working on software engineering projects. One observation is that students are able to work more effectively in a team environment than past students. Another observation is that students are at least as technically capable as previous students. Further, feedback from students who have undertaken the PBL version of this unit indicates they feel more confident in a new teamwork environment and that they find the research skills learnt in this unit are applicable in follow-on units. One of the author's and a co-researcher have recently received a grant to conduct a detailed review of the results of this project. They will not only compare results students achieved in follow-on units between the pre and post PBL

cohorts, but will also conduct surveys and interviews with employers of students from these cohorts. The results of this research will be published when completed.

Hattie (1999) used effect size to measure the effect of an innovation compared to some other innovation (as distinct from no innovation) on student achievement in primary and secondary schools. Examining some 300-400,000 effects from 510 meta-analyses, Hattie (2004) reported that 41 PBL innovations had a mean effect size of 0.12, i.e. 0.12 of a standard deviation (note, the effect size of innovations in schooling is 0.4). That is to say, implementing PBL alone, without changing the curriculum design and teaching such as we did had very little impact on student achievement and was much worse than a 'standard innovation'.

3.2 Pedagogical Changes

In introducing PBL, we changed the curriculum, the teaching method, introduced an entirely new teaching team to the unit, and also provided the new team with a development programme in PBL. From a pedagogical perspective all these factors appear to have contributed to the success of our implementation of PBL.

Firstly, we severely stripped the curriculum by reducing the content to include only what students needed to know and be able to do at the end of the unit. Anything considered redundant, such as material more appropriately and currently also taught in later units, was removed from this unit. The material remaining was organised into three or four coherent learning objectives for each PBL problem. Real-world scenarios were then developed around the objectives. Scoping and writing successful triggers is a challenge and requires careful attention.

Secondly the real-world scenarios were designed to be relevant and interesting to first-year students. The products that students developed as part of the curriculum were ones that engaged them and that would be useful in different sectors of society and would therefore give them a very real sense of accomplishment.

The new teaching team, with its fresh enthusiasm, may have had some impact on the reduction of the failure rate and improved results. But, previous changes to teaching teams in any unit had not made such a significant impact. From this it seems clear that it was the new teaching approach taken in the unit that led to the significant improvement in student learning.

We also introduced development programmes for tutors to become familiar with PBL and to give them ongoing support and development during the semester. The main objectives of the initial tutoring programme were to introduce tutors to PBL and the PBL process, help them develop their ability as facilitators, and forewarn them of the common difficulties experienced by both tutors and students and provide them with an opportunity to work through the issues in their tutor peer group prior to the start of semester. The on-going programme for tutor development provided tutors with weekly support in both content and managing successful PBL tutorials. The latter included managing student participation in the tutorial process, and giving meaningful feedback to students on their performance.

Additionally, since 2003 some other units within the faculty at QUT have seen similar improvements. These units have also undergone major changes to their teaching approaches, and adopted many aspects pioneered in this unit.

3.3 Reduced Isolation

There were other unexpected but welcome outcomes from the PBL small group work. Recent surveys indicate that many students work up to 15 hours a week in paid employment with 18% working 21 hours or more (McInnis and Hartley 2002). In addition, most first-year students at our university feel isolated in their studies (Hart and Stone 2002). Our first-year students, who liked group work, commented that it was one of the three best things about the unit. They met other students and developed meaningful friendships which carried over into other units. This was particularly significant in a large first-year class in an era where students spend relatively little time on campus due to life and work commitments.

3.4 Group Work

A large proportion of the problems students encountered with group work in the earlier implementations were due to unequal sharing of workload. In some groups, a few members worked harder than others, while some made little or no contribution to the team at all. We alleviated this problem by introducing a marking scheme that rewarded both group and individual work. In this scheme, team members shared identical marks for those assessment items that were completed as a group (for example design documentation, system test plans, functionalities achieved etc.) and received individual marks for those items they completed individually (code quality, unit test plans, reflective statements etc.). In addition marks were allocated to compensate hard-working team members who's marks were unfairly penalized by the poor performance of other team members. Or, when justified, negligent team member(s) had their marks reduced in one or more categories, or zeroed in extreme cases. This solution greatly alleviated the frequency of such problems occurring, presumably because participating team members felt there was some recourse "built-in" to the marking procedure.

Another group work problem was identified. Although all members participated (by their analysis equally) in the final solution, the coding work was sometimes done by only one or two group members with previous coding experience. This 'specialisation' circumvented one of the major learning objectives of the unit: that all students became competent intermediate-level programmers. To overcome this, we introduced an assessment condition to ensure that each team member provided evidence that they made an equitable contribution to the writing of code required to complete the assignment. To ensure this, each individual's code needed to be accompanied by a signed statement, in which the team member declared that he or she was its sole author.

An additional issue was identified – some groups worked so well together that they extended their systems far beyond that which was required. Although some came up with exciting and highly imaginative designs, others demonstrated a problem with prioritising by placing more emphasis on the requirements they enjoyed implementing, rather than focusing on what the "client" asked for (via the functional specification). Some tutors had difficulty reigning in groups to concentrate on the actual depth and breadth of the problem.

3.5 The Use of 4SAT

Getting students to recognize factors of successful teamwork in their own behaviour continued to be an issue. Although the 4SAT instrument focused students' attention on their interpersonal interactions and participation in tutorials, some students were either slow or failed to recognize their own or their group's unsatisfactory behaviour(s). In the first iteration, we piloted the PBL problems with a small class of 26 students and used an expert-guided playback of video recordings of tutorials to assist students in observing their behaviour (Adams et al. 2001). Although successful, this strategy was not practical for large classes. To provide an 'independent observer' we defined an additional role for students as a process observer for the PBL tutorial. However, students rarely acknowledged unsatisfactory attainment of any of the instrument's criteria. This self-rating of participation is an aspect that needs to be further developed, and has implications for student self-rating of graduate capabilities for student portfolios.

3.6 Improved Satisfaction

We evaluated student satisfaction with the unit through both informal comments from students to tutors and by formal interviews with students at the end of semester. Both informally and formally the majority of students commented that they were confident they understood the material and that they enjoyed working on the problems. Brighter students started to talk about their approaches to learning and problem solving. Even students who were less enthusiastic about the PBL approach in early years of their degree programme, in later years of study reported that they were realising the benefits of the PBL approach.

Indirectly and directly, most tutors reported that they enjoyed their PBL tutor roles in the unit. We noted that tutors were requesting to tutor in the unit because they had heard it was interesting and enjoyable. Tutors also reported that they wished they had had the opportunity to study in a PBL curriculum when they were studying. One or two tutors reported discomfort with the facilitation role.

4 Conclusion

In this paper we describe the cases we developed and insights gained in implementing a successful PBL curriculum in intermediate-level Java programming. The cases were a significant improvement on the former, traditional lecture and laboratory classes in many ways. Students were quite highly motivated by the real-world scenarios and enjoyed developing the software products that had real-world application.

Graduate capabilities surfaced as a by-product of implementing the real-world scenarios. Although interpersonal communication, so important in the IT work environment, was reinforced through specific strategies (use of Step 2b of the 4SAT), further work in this area is required to make significant headway. Quite unexpectedly, as a result of the friendships established in the small group work in this unit. students reported experiencing reduced isolation in other units of their first year studies.

We considered requests from tutors to specifically tutor in this unit and improved attitudes towards the teaching method by some of the full-time staff required to tutor in the unit as signs of a breakthrough.

Our implementation of PBL was successful in cutting the failure rate for intermediate-level Java programming to well below the national average. The essential pedagogical features included making some hard-headed decisions about the content by cutting the learning objectives to a few significant ones; developing cases that had a real-world feel; restructuring the large lecture and relatively large laboratory classes to small group tutorials in which students were required to interact with each other and the coursework material; introducing a standardised, tailored PBL tutorial process to assist students and tutors to work through the cases; and providing initial and on-going development for lecturers and tutors.

5 References

Adams, M., Clarke, S. & Thomas, R. (2001): Developing Graduate Capabilities through PBL. *Proceedings of the 3rd Asia Pacific Conference on Problem Based Learning*, Rockhampton, Qld, Australia.

Barrows, H. S. (1989): *The Tutorial Process*. Revised edition. Springfield, Southern Illinois University School of Medicine.

Bentley, J., Lowry, G., & Sandy, G. (1999): Problem Based Learning in Information Systems Education: Preparing Students for Professional Practice in a Project Environment. Presented at the *SAICSIT'99*, Hartebeespoort, South Africa.

Bloom, B. S. (Ed). (1964): *Taxonomy of educational objectives: the classification of educational goals.* New York, Longman.

Bowden, J. and Marton, F. (1999): *The University of Learning*. London, Kogan Page.

Clark, D. C. (2001): Lost in the Mêlée. In *Problem-Based Learning: Case studies, experience and practice*. 34-39. SCHWARTZ, P., MENIN, S. and WEBB, G. (eds). London, Kogan Page.

Clarke, S., Thomas, R. and Adams, M. (2001): Model of Thinking in the PBL Process: Comparison of Medicine and Information Technology. *Proceedings of the 3rd Asia Pacific Conference on Problem Based Learning*, Rockhampton, Qld, Australia.

DEST (2002): Department of Education Science and Technology. Australian Federal Government website. http://www.dest.gov.au/tenfields/science/under.html#COMPUTER SCIENCE. Accessed 11 February 2003.

Hart, G. and Stone, T. (2002): Conversations with students: The outcomes of focus groups with QUT students. Proceedings of the 2002 Annual International Conference of the Higher Education Research and Development Society of Australasia (HERDSA).

Hattie J. (1999): Influences on Student Learning; Inaugural Lecture: Professor of Education, University of Auckland. Source: http://www.arts.auckland.ac.nz/staff/index.cfm?P=5049 Accessed 1 Nov, 2004.

Hattie, J. (2004). Research in Health Professional Education: The nature of research. One-day workshop: Research Methods in Health Professional Education, 8th September, 2004, Faculty of Medical and Health Sciences, University of Auckland.

McInnis, C. and Hartley, R. (2002): *Managing Study and Work*. http://www.dest.gov.au/highered/eippubs.htm #02_06. Accessed 11 February 2003.

Neilsen, A.C. (2000): Employer Satisfaction with Graduate Skills: Research Report. *Evaluations and Investigation Programme*. Canberra: DETYA, Higher Education Division. http://www.dest.gov.au/archive/highered/eippubs/eip99-7/eip99_7pdf.pdf. Accessed 11 February 2003.

Newble, D. I. & Clarke, R. M. (1986): The approaches to learning of students in a traditional and in an innovative problem-based learning medical school. *Medical Education*. **20**:267-273.

Ramsden, P. (1992): *Learning to Teach in Higher Education*. London, Routledge.

Searle, N. and Clarke, S. (2000): *Steps in writing a good problem*. Unpublished document based on websites from the University of Adelaide, the University of Newcastle and the University of Delaware.

Turner, R. & Lowry, G. (1999): The Compleat Business Information Systems Graduate: What Students Think Employers Want and What Employers Say They Want in New Graduates. *Presented at the Pan-Pacific Business Association Conference XVI, Fiji*.

Zimitat, C. & Alexander, H. (1999): The 4 Step Assessment Task (4SAT). *Integrity, Innovation & Integration*, **4**:692-697.