Developing the Software Engineering Team

James M. Hogan and Richard Thomas

Centre for Information Technology Innovation
Queensland University of Technology
GPO Box 2434, Brisbane, Qld. 4001

j.hogan@qut.edu.au and r.thomas@qut.edu.au

Abstract

Teamwork is often considered one of the most important "generic skills" that we can provide to graduates entering the information technology profession. Often though, through the rush of covering important content, we short change our students by giving them limited opportunities to learn how to work effectively in teams. Students also often complain that although they are expected to work in teams on projects, they are never given any advice or guidance on how to work in a team (Hart and Stone, 2002). Or, if they are given guidance, it is often from a business perspective that students find difficult to integrate into their software development practice.

In this paper we discuss a course-spanning initiative to help students learn teamwork skills. This initiative starts in first year by emphasising a core set of skills directly related to working in teams. These skills are applied in small software development teams, with close tutor supervision focusing students on teamwork rather than upon individual software development. The theme then continues into second and third year where the growing sophistication of the students' teamwork skills is complemented by increasing their independence and requiring students to apply a professional software engineering process as a development team. By the end of their course students are then in a position to work together effectively solving complex problems for a real industry client.

Keywords: teamwork, software engineering, problem based learning

1. Introduction

To the uninitiated – and especially to the enthusiastic amateur – software development is often seen as an individual pursuit involving long hours of direct interaction with a computer. This image is reinforced by introductory programming subjects in both secondary and tertiary education, with students assigned simple programming projects to be completed entirely on their own. Indeed it is common to punish group collaboration as a form of plagiarism. In contrast, professional practice is dominated by the team environment: it is the team that enables productive software development. Many projects are too large to be completed by an individual; some of the largest projects have their cumulative effort measured in thousands of person years. Even in small projects it is normal practice to use a team to leverage the (hopefully) complementary set of skills that a group of developers may bring to a project.

In consequence, the ability of each person to collaborate effectively within a team may govern the viability of a software house. One person working at cross purposes to the team or undermining its cohesion can dramatically reduce its effectiveness. To better prepare our students for professional practice, the Faculty of Information Technology (FIT) at QUT has introduced a range of initiatives in teaching and learning centred upon teamwork. These innovations commence in the second semester of first year and continue throughout the three year undergraduate degree, progressing from carefully controlled and simulated practice to direct exposure to a professional environment, with only partial mitigation of its risks and variability.

In this paper we consider the progression of team development within two complementary frameworks: the educational blueprint of problem based learning (PBL), and the professional guidance provided by a modern software engineering process – here the Real World Software Process (RWSP) (Hogan et. al. 2002), developed in-house specifically for this role. Each approach has a role to play throughout the degree, but the balance between them shifts subtly as the course progresses, reflecting the growing social and professional maturity of the students.

Within PBL, students learn to take responsibility for their own learning within a carefully structured team environment. Independent research and presentation skills reinforce the workings of the team, but the assigned task cannot be completed without successful interaction with other team members, and – critically – reliance upon their contributions. Process-oriented project teams build upon the skills acquired through PBL, working collaboratively within the novel framework of a defined software engineering process to produce a significant piece of working software. Elements of this process appear within the first year offerings, but it is only in the more advanced subjects that it takes a central role. Students rely upon the developing team environment to support their understanding of the process; and the process, in turn, to

support the development of a professional and effective team[2].

Each of these aspects of our curriculum is geared toward simulating – and in the later subjects, realising – a healthy, well-functioning team environment. While there has been robust debate about the precise characteristics of the most successful software development teams, there is broad agreement (as is shown by even the most casual review of software engineering textbooks and course descriptions) that certain key areas must be addressed, principally communication and time management. These selections are somewhat brutally reinforced by even a casual exploration of the literature of large-scale project failure, but they are also important factors when schedule and cost over-runs and unacceptable defect rates appear on a smaller scale. Thus, development of skills in these areas forms the core of our approach, and much of this paper concerns our attempts to integrate this material within a broader practical context, and to incorporate the related notions of leadership and decision making, self-directed learning, and a capacity for review and reflection.

These issues are explored in some detail in section two, leading us naturally to consider their instantiation within the information technology curriculum – including a detailed examination of the PBL and process-centric initiatives we have undertaken within the Faculty (section three). The success of these approaches and their effect on student learning is considered in section four, and we conclude with some discussion of the future and our plans for longer term studies of the learning outcomes.

2. Aspects of Teamwork

It is self-evident that an enormous range of factors may influence the effectiveness of a professional team, and that only a fraction of these may be addressed within an undergraduate environment. Our initiatives are centred around communication and time management, but the approach is more broadly drawn, incorporating the complementary skills of decision making and leadership, and building the student's capacity for self-directed learning and reflection. Such generic skills are necessary for successful teamwork in many fields, but they are critical in the software development team.

Effective communication is imperative in any team-oriented enterprise, and flawed communication has emerged as a root-cause in many of the documented case studies of industry failure (McConnell, 1996). A key factor in effective communication is the recognition that success requires "two-way" interaction. Transmission of an idea is not sufficient: it must be received and understood accurately, and the necessary confirmation is often missing from informal team meetings. Miscommunication can be the result of the receiver not understanding the message content, or being unwilling to accept it from the particular source. Egoless programming is an ideal seldom achieved in practice, but team members must be able to express concerns about the progress of a project, and have those concerns considered professionally. Indeed, McConnell concludes that many of the problems which beset development teams are the result of wishful thinking maintained in the face of mounting evidence, and frank communication is critical in challenging this comfortable fantasy.

In each case, communication failure may have significant consequences and team members must learn to communicate their ideas successfully. These skills are established early through the PBL approach and through formalised support for team meetings as part of the software engineering process.

Successful communication, together with a willingness to take responsibility for a defined component of a project, is an essential prerequisite for managing team activities and working within a project schedule. Time management is crucial for team success, and the foundations are laid in our approach through a mix of external control and internal flexibility. While PBL teams must work within a rigid, externally imposed schedule, students must themselves allocate tasks across their team to ensure deadlines are met. In later subjects, students are given progressively more responsibility for creating their own task estimates and project schedules, and for managing the conflicts which arise as slippages occur. Yet effective time management, like communication, cannot develop without adequate scaffolding, here provided through formal guidance on project planning and time estimation, and templates for documenting assigned tasks and the success of team members in completing them on time.

Slippages in schedule are not the only source of conflict among developers, and concerns over the allocation of component tasks are especially prevalent among student teams, in which no line management accountability has been established. Successful decision making within a project team requires both leadership and the willingness of each team member to support the decision and to take responsibility for their assigned component. While a formal command structure may not be appropriate in the professional environment, leaderless teams lack direction and waste time coming to decisions. The leader need not be a strong driving force with a large influence; effective leadership may come from a coordinator or mediator who helps the team to accomplish tasks in an efficient manner. Regardless of the style, however, our experience has shown that teams which include a leader "personality" type tend to be more successful and harmonious than those which do not (Thomas, 1999).

The existence of a leader should not allow the remaining developers to discard all responsibility for the team direction. Even with a leader in the team it is still important for each member to understand and agree to a

[2] All genuinely professional software developers guide their activities – from the initial interactions with the client through to the maintenance of the system after delivery – according to a defined software engineering process. Healthy interactions of the type described require a commitment from the team to work according to the process, and to improve their performance through reflection and revision for subsequent projects. Such commitments seldom emerge if there is a poor match between the process and the style and experience level of the team. These issues are considered later as we examine process infrastructure as a guide for team development.

specific decision making style. This provides an avenue for any developer to solicit a decision, and helps to reinforce support for unpopular decisions once they are made. Both formal and informal leadership roles may exist within a team. In most subjects involving teamwork, formal leadership roles are defined and allocated to students. Tutorial staff may provide advice and monitor progress – although in our programme the degree of supervision declines markedly after first year – but the decisions and their consequences remain the responsibility of the students. In a number of cases we have also worked with students to help them identify and use these informal leadership "personalities" within the team[3].

Ideally, the diverse range of professional experience within the development team will inform the allocation of tasks, and limit the risk of disputes among its members. Inevitably, however, a precise match between task and experience is unlikely. Given the dynamic nature of the IT industry and the consequent limited shelf life of technical experience, all successful developers must be able to recognise their deficiencies and must have the facility to acquire new technical knowledge rapidly and independently. Arguably more so than in any other discipline, lifelong learning skills are imperative for the IT professional who wishes to remain employable. The problem based learning approach within first year is designed principally to establish self-directed learning skills within the student body, but its mechanisms offer an additional advantage for the development of teamwork. A central tenet of the PBL methodology is that learning is greatly enhanced when the student has an additional role as teacher, the newly established specialist communicating new-found knowledge to the remainder of the group. This process has deep echoes in the professional development team, as novel software technologies are absorbed as preparation for new projects, and it is some measure of the success of the first year preparation that student developers recognise this responsibility to their team mates, providing ad hoc informal tutorials as appropriate.

Problem based learning also provides the groundwork for the most important skill that a developer can bring to the professional team: the ability to reflect upon performance during a project – upon their own performance as an individual developer, upon their individual contribution to the cohesiveness of the group, and upon the quality of the team work and the process which governs their approach.

Reflection is an important aspect of learning; at the individual level it is necessary for students to reflect on newly acquired knowledge and to integrate it within a larger cognitive framework. Reflection upon their learning activities may similarly improve and expand their capacity for learning, and its extension from the classroom to professional practice is a significant factor in managing a constantly changing environment.

[3] Defined roles within the team appear to be an important aspect of the student experience, whether or not they are associated directly with leadership. We shall return to this issue later in the paper.

To highlight the importance of reflection, all subjects that involve teamwork require students to reflect on the team and process. In their first PBL subject, students are expected to devote a significant amount of time and effort to reflection each week. In later subjects, the formal requirements are more limited, but student teams must minimally reflect upon their teamwork and their process at the end of each project cycle – an approach which directly mirrors successful industry practice.

3. Teamwork in FIT

Group projects have long been common practice in software engineering degree programmes (Shaw and Tomayko, 1991), typically being scheduled towards the end of the course and treated as a "capstone" project. This approach is reflected in both the ACM/IEEE joint model curriculum in computer science and software engineering (Chang et al, 2001 and LeBlanc et al, 2004) and in the model curriculum devised by the Software Engineering Institute (Bagert et al, 1999). While teamwork skills are usually acknowledged as important learning objectives for group projects, in many cases these outcomes are assumed to emerge automatically, and little direct emphasis is placed on ensuring that the skills are developed.

Traditional capstone projects have often followed a waterfall lifecycle model, proceeding in turn through each of the standard activities of the development process (see for example Sommerville, 2001). The single pass inherent in this model focuses teaching and student attention on the process, providing no formal learning opportunities for teamwork skills. Newer approaches such as the Agile Methodologies (Agile, 2001) rely upon frequent iterations of a lightweight process, with reflection and review at the end of each cycle a key aspect of the methodology. Iterative approaches thus provide a ready-made opportunity for teamwork to be addressed directly.

Yet it is our view that successful reflection requires some experience and understanding of the dynamics of the team environment, and that more sustainable outcomes will result if students enter the development team adequately equipped. In our programme, this foundation is provided at the first year level through PBL, with the skills of communication, reflection and self-directed learning contributing to the success of the software project, and being reinforced and extended through experience. We now discuss this strategy in some detail, examining particularly the role of the PBL process and its linkages to subsequent development team practice.

Explicit support for teamwork and for the development of related skills has been part of FIT's core software engineering project subject(s) for more than a decade, with ongoing efforts to improve the quality of the material offered and its relevance to industry practice. More recently, these efforts have taken the form of a large-scale initiative known as the Real World Software (RWS) Project (Hogan et. al., 2003), which took a structured approach to developing teamwork throughout the student's undergraduate experience.

Through the RWS project, teamwork was introduced in a first year laboratory unit within the PBL framework; in

subsequent revisions of the first year programme, this has been extended to encompass three subjects. In each of these offerings, teamwork skills are among the primary learning objectives. To support this emphasis, students are provided with a team process to follow and are given extensive tutorial support. Each group (usually five to eight students) is assigned a tutor who works with the team for one to two hours per week for the entire semester, focussing upon team and process issues.

In second year, students work within a five-person development team on their first genuine software engineering project (Software Engineering Principles or SEP). The key learning objectives are here centred on software engineering theory and its small-scale application, but there is a strong secondary objective to enhance teamwork skills. Final year students again work on a software engineering group project, with the additional complication of a 'live' external client (Advanced Programming Laboratory or APL). While the primary learning objectives are again related to the application of theoretical content, well-developed teamwork skills are essential if the client's requirements are to be met, and interactions managed successfully. We consider each of these stages in turn.

3.1 Problem Based Learning

One of the aims in implementing PBL was to better prepare students for professional practice by shifting the focus of education from teaching to learning (Bowden & Marton, 1999). PBL has a role in developing the graduate capabilities of teamwork and communication skills (Lovie-Kitchin, 1998; Greening, 1998; Petersen, 1997; Bentley, 2000). However, we recognised that students cannot be expected to develop these skills by osmosis: teaching is needed to encourage their development (Bowden & Marton, 1999).

Following the advice of Boud and Felletti (1998), real-world "problems" or scenarios were used as a "stimulus and focus for student activity". We also recognised the importance of providing a framework to enable students to work through PBL problems, and modified the Maastricht 7-jump model (Schmidt, 1983) to fit our environment. Here, PBL problems span several weeks – rather than the more intensive model usually employed (Rideout & Caprio, 2001) – with one two-hour tutorial each week. In the first week of a new problem, students wrap up the previous exercise and commence the new activity within the one session.

3.1.1 Note Initial Reactions

Student teams begin each PBL problem through an introductory "trigger", with their first activity being to note the reaction of their team to the problem. This step was added to the process to support the teamwork objectives, assisting team members to understand each others' emotional responses to a particular problem, and to help build empathy within the team. This is especially important if certain individuals have had unsuccessful "real life" experiences in a related problem domain.

3.1.2 Analyse Problem

The team's analysis of the problem begins with an attempt to clarify terms and concepts. Team members are able to learn from each other as they explain particular aspects of the problem that others are having difficulty comprehending. Students are thus led into the role of "teacher", an important step in their development as self-directed learners and one which builds trust within the team.

3.1.3 Activate Prior Knowledge

The *activating prior knowledge* step helps students to learn the value of each other's contributions. Each team member must identify aspects of their knowledge and experience which may be used to solve parts of the problem.

3.1.4 Formulate Learning Objectives

Continuing the theme of self-directed learning, the team identifies learning objectives aligned with aspects of the problem they cannot solve. Team members must then allocate learning objectives amongst themselves through negotiation, supporting the development of communication and time management skills. Subsequently, students must take responsibility for their assigned tasks, and ensure that they are completed before the next meeting.

3.1.5 Research Learning Objectives

Successful team work requires that students recognise the importance of each individual's contribution. Learning objectives are addressed at an individual level, but the knowledge is required by the entire team. Students thus learn that *all* team members contribute to successful team performance. If even one team member fails to complete assigned tasks, progress of the entire team may be jeopardised. Reliability of team members is essential if the team is to function efficiently; otherwise more conscientious team members will end up duplicating work or over-working to make up for the failings of others.

3.1.6 Report Back

Reporting back supports development of communication skills, as students explain the results of their research to the rest of the team. Important new information must be communicated if it is to be applied in solving the overall problem. Students are encouraged to use handouts and walk throughs of program code to support their explanations. This has the additional benefit of improving written communication skills and providing the team with a record of the material.

3.1.7 Analyse Additional Issues

Analysis of additional issues allows the team to work as an "organic whole", synthesising new material and planning its application to the problem. This improves the student's grasp of new knowledge and facilitates project scheduling, forcing the team to consider their progress toward the overall objective. At the completion of this step, the team should either be in a position to solve the

problem, which would become their next task, or they are given an additional trigger that leads them back to the first step in the framework.

3.1.8 Wrap Up

The final step of wrapping up the problem requires students to demonstrate their solution to other teams in the same tutorial session, exercising communication skills and building team cohesion as they receive peer comments. Significantly, students are forced to examine the quality of their teamwork during the problem – providing an explicit opportunity to deal with issues within the team and to optimise team processes.

3.1.9 Framework Discussion

It was intended that teamwork skills be developed through the learning opportunities provided by the framework, as described above. But, as implied by Bowden and Marton (1999), opportunities do not guarantee that learning will take place – particularly in a large first year class with a very diverse cohort of students. Thus, each team was allocated a tutor to guide them through the process and closely monitor their progress. Here, the tutor is less a content expert than a facilitator who acts to develop teamwork and self-directed learning. This close interaction between tutors and teams ensured that the vast majority of students improved their teamwork skills during the subject (Adams et al, 2001).

3.2 Software Engineering Teams – *Software Engineering Principles (SEP)* and *Advanced Programming Laboratory (APL).*

By the end of first year, the student's professional armoury is well-established, with the fundamentals of teamwork acquired within PBL. Subsequent team projects are thus less directed, our focus being to support young professionals as they improve their skills. In particular, the close supervision of the PBL tutor is replaced by the lighter touch of the Development Manager, here a tutor acting as a professional line manager responsible for several teams. The apparent supervision deficit is readily filled by student leadership and the availability of accessible tutorial and process material to support team processes.

The core of this material is provided within the Real World Software Process (Hogan et. al., 2002), an in-house Agile development methodology introduced as part of the RWS Project. Agile methodologies have received considerable attention within the industry, but the critical driver behind RWSP was our belief that the lightweight process model offers enormous opportunities for team-based education.

Agile methodologies are centred upon *rapid iteration*, in which the entire software development process is repeated until the desired functionality is delivered[4]. These short cycles are in marked contrast to the traditional view, in which a single development cycle is completed over the lifetime of the project. In the educational context, a repeated lightweight process cycle involves substantially more work for the academic staff and a tight schedule for the students, but it allows a number of important advantages:

- Students gain greater familiarity with the process, and are faced with more realistic challenges in the team environment – particularly in the need to manage at least two software product releases.
- Students have an opportunity for reflection upon their effectiveness in using the process and in assessing its usefulness as a guide for their work. This reflection and discussion of potential improvements to the process is an essential aspect of process and developer maturity.

It is this explicit allowance and mandate for reflection that allows our development projects to reinforce the skills acquired during first year, but once again the activity must be adequately supported. Within the professional software engineering community, the quest for the perfect development process has been tempered by the recognition that the process is effective only if it is used, and there are abundant anecdotes whose central theme is the expensive, carefully constructed and ultimately useless process support material which lies on the bookshelves, unused during actual development. Such difficulties are especially pronounced for inexperienced teams, of which student software engineers are the archetypal example, and our process material is distinguished by the degree of tutorial support which is integrated into the process description.

Teamwork support within RWSP is explicit, encompassing templates for time management and for meetings and their resultant actions, together with extensive guidance upon the activity and the use of the appropriate templates[5]. Our approach is similar in some respects to the Personal Software Process (Humphrey, 1994), albeit with simplified data collection and no direct attempt to measure productivity. Our focus remains the coherence of the team, rather than the determination of effort, and the process material allows a degree of independence that would not otherwise be possible.

Coherence is additionally fostered through careful attention to team selection, bounding the risk of conflict by ensuring that the most experienced and technically able developers are distributed across the groups, rather than allowing their concentration through self-selection[6]. However, student comment suggests that this approach is

[4] Indeed, in extreme programming (Wells, 2003), the best known agile approach, such a cycle may occupy less than two weeks of full time development.

[5] In some respects this material is a natural extension of the support that the Faculty has provided to undergraduate software engineering students for many years, but the information integrated within the process is markedly more elaborate than in the past, and covers a wide variety of issues – ranging in sophistication from single paragraph definitions to extended guidance for particularly troublesome process activities. This material may be viewed in situ at:

http://www.fit.qut.edu.au/~rwsp

[6] Much of the present approach is due to Dr. Sam Stainsby.

insufficient of itself, and teams are more likely to be harmonious if there is a greater concentration on the roles performed within the team, so that each developer is seen to provide a valuable service to the project. This naturally reinforces the lessons of 3.1.5, and the value of each contribution.

RWSP is employed initially within SEP, and student understanding of the development process and the importance of teamwork is well-established by the conclusion of their second development iteration. Reflection within the team is required explicitly – as part of the assessment requirements – and implicitly, through the need for refinement of failed strategies from the first pass. The extent to which individual contributions are valued is apparent from peer assessment, with poor scores extremely rare and usually well justified[7]. Skills thus developed are a firm basis for extension to the more substantial and challenging tasks of APL.

Modern software engineering education is increasingly driven by an expectation that projects should reflect industry best practice, and utilise state of the art software technologies. The use of development projects in conjunction with an industry partner provides a direct response to these challenges, enhancing student and team motivation through the nature of the project and the desire to perform professionally in front of a potential employer; significantly, employers are not impressed by dysfunctional development teams.

As one might expect with such strong industry linkages, there may be radical change in the software technologies employed from semester to semester, providing an immediate test for the self-directed learning skills developed across previous subjects – particularly in the degree to which the team is able to facilitate learning of new technical material. The preparation provided by PBL corresponds to commercial practice, in which new technologies are absorbed and communicated across the whole of the development team.

Ultimately APL offers the opposite extreme from first year PBL, with teamwork so ingrained that its development is the province of the students. Experienced tutorial staff will discuss process issues regularly with their teams, but intervention is extremely rare, ad hoc clarification of issues at the request of the students being the most useful contribution. The APL cohort is dominated by high achievers, and their skills require polish rather than remoulding, and this is supplied as necessary from academic and industry staff alike.

Yet the complexity of the project provides a ruthless examination for the team, with a successful outcome critically dependent upon:

- Adequate communication between the team and the industry partner to capture user requirements and clarify them over time;
- The ability of specialists within the team to acquire new technical skills and to educate their team-mates; and
- The ability of team members to rely upon the timely contributions of others.

It is our experience that poor performance in *any* of these areas is sufficient to ensure a substandard project.

4. Student Experience and Reflection

Student experiences across our subjects reflect a gradual development of communication and time management skills, and an ability to work cohesively within a team – although this too is more pronounced with progression through the degree. Marked improvements in the capacity for self-directed learning are reported from the end of first year, and the importance of these skills noted in subsequent offerings. Our discussion of student experience is based upon (Hogan et. al., 2003), which drew upon focus groups and written responses from each of the student cohorts, and an external review of the PBL offering by a prominent researcher[8].

The introduction of a PBL subject within the first year programme was not without its teething troubles, and these are considered at length elsewhere. However, by the third offering of the subject the approach was proving successful, and subsequent experience has confirmed the optimism of Farmer's 2001 report. Much of the initial student dissatisfaction centred upon the translation of group assessment into individual grades – an issue particularly troublesome to some high achieving students. This was subsequently addressed through a revision to the assessment scheme, but the lack of contribution from some team members has remained the principal source of student concern:

> *"Some group members have lower expectations and commitment, 'shirkers', group 'slackers'."*

In relation to teamwork, however, student acceptance of the approach and its rationale was markedly better:

> *"It is about teamwork … also about learning programming…I have changed my views about [the importance of] teamwork."*
>
> *"[I have learned] how to start from scratch instead of being spoon fed."*
>
> *"It has been a good learning process for the group especially about professional responsibility issues."*
>
> *"I have been so involved especially in making decisions, in group management and project management. PBL is useful after graduation."*

[7] Individual contributions are assessed by a combination of two equally weighted scores, one from the tutor and the other the mean of assessments from peers. Peer assessment is controlled in the same manner as Olympic diving or gymnastics, with the outliers discarded. Thus, a low peer score is almost certain to reflect a particularly poor contribution.

[8] The review was conducted by Associate Professor Liz Farmer of Flinders University, and the focus groups and resulting reports by Naomi Searle and Jenny Reye.

"Group environment was excellent, got to learn from others, teamwork is a necessity. Meeting new people and creating networks."

This appreciation of teamwork skills and their importance upon graduation is reflected in reports from subsequent subjects.

Students of SEP noted the importance of communication and the need for trust between team members, with compromise and negotiated task allocation essential to team harmony and a successful outcome. In contrast to the PBL experience, few team members failed to contribute adequately.

Not unexpectedly, students reported only gradual development of their time management skills rather than an immediate, intensive improvement. Here the principal aids were seen to be an unforgiving project schedule with extremely regular deliverables, and the PSP-like work logs and meeting planners which allowed greater tracking of progress.

At the process level, the responses confirmed our intuition about the educational advantages of iterative development, with familiarity and experience reinforcing and supplementing the lessons of the initial phase. A number of students commented on the role of reflection in improving technical practice and team performance.

Within APL, student reports have focused upon the motivating role of the industry linkages:

"Students enjoyed and were motivated by the real life project and industry partnership. All of the students agreed that it was one of the most interesting units they had done at university, and despite the difficulties, they enjoyed the subject. Students felt that it prepared them well for work in the IT industry…"

and upon concerns over underperforming team members. While limited in scope, we believe these latter concerns reflect a clash between the professional culture which dominates the cohort and those whose team skills and commitment have not matured to the same extent.

In closing our discussion, we acknowledge with some unease our reliance upon student reports of skill development without some deeper examination of the factors underpinning success. Direct assessment of teamwork and the supporting skills is difficult of itself, and still more difficult to isolate from content influences. Within the controlled PBL environment it is possible to capture skill development through the 4SAT instrument (Zimitat & Alexander, 1999), and so we are re-assured by the alignment of 4SAT results and student reports. However, 4SAT is not applicable within the development team subjects, and even student project marks must reflect a multitude of performance criteria.

Ultimately, the long term impact of our work can only be assessed once student cohorts have progressed successfully into the workforce, and been given appropriate follow up. However informed the student body in making judgments, there can be no substitute for the reflection which accompanies actual practice. Until this final assessment is completed, there remains a modest danger that we have been too successful in providing a reinforcing context for the initiatives, and that the student reports unduly reflect *our* worldview. Yet, this danger is mitigated by the sophistication of our student body, their exposure to numerous additional information sources and in some cases, there experience as quasi-professional employees within the industry. We remain confident that our approach has contributed to the development of outstanding professional teams.

5. References

Adams, M., Clarke, S. and Thomas, R. (2001): Developing Graduate Capabilities Through PBL. *Proceedings of the Third Asia-Pacific Conference on Problem Based Learning*. PROBLARC, 2001.

The Agile Alliance (2001): *Manifesto for Agile Software Development*. http://www.agilemanifesto.org/. Accessed 26 August 2004.

Bagert, D., Hilburn, T., Hislop, G., Lutz, M., McCracken, M. and Mengel, S. (1999): Guidelines for software engineering education version 1.0. *Technical Report CMU/SEI-99-TR-032*.

Bentley, J. (2000): Teaching Introduction to Business Systems Development: A Problem Based Learning Approach (Working Paper). Melbourne: Victoria University of Technology.

Boud, D. and Feletti, G. (1998): *The Challenge of Problem-Based Learning*. London, Kogan Page.

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

Chang, C., et al. (2001): *Computing Curricula – Computer Science Volume*, http://www.computer.org/education/cc2001/final/index.htm. Accessed 26 August 2004.

Greening, T. (1998): Scaffolding for Success in PBL. *Medical Education Online*, **3**(4). http://www.med-ed-online.org/f0000012.htm. Accessed 26 August 2004.

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)*.

Hogan, J., Smith, G. and Thomas, R. (2002): The Real World Software Process. *Proceedings of the Sixth Asia-Pacific Software Engineering Conference (APSEC 2002)*: 366-375.

Hogan, J., Thomas, R., Clarke, S., Smith, G., Adams, M. and Ho-Stuart, C. (2003): The Real World Software Project: Improving the professional education of software developers through Problem Based Learning, self-guiding Software Engineering Processes and Software Quality Tools. *Technical Report, Faculty of Information Technology, QUT*.

Humphrey, W. (1994): *A Discipline for Software Engineering*. Boston, Addison Wesley.

LeBlanc, R., Sobel, A., et al. (2004): *Computing Curricula – Software Engineering Volume*. http://sites.computer.org/ccse/. Accessed 16 July 2004.

Lovie Kitchin, J. (1998): Problem-Based Learning in Optometry. In *The Challenge of Problem-Based Learning*. 203-210. BOUD and FELLETTI (eds). London, Kogan Page.

McConnell, S. (1996): *A Case Study of Classic Mistakes*. http://www.stevemcconnell.com/rdmistak.htm. Accessed 12 September 2003.

Peterson, M. (1997): Skills to Enhance Problem Based Learning. *Medical Education Online*, **2**(3). http://www.med-ed-online.org/f0000009.htm. Accessed 26 August 2004.

Rideout, E. and Carpio, B. (2001): The PBL Model of Nursing Education. In *Transforming Nursing Education Through Problem-Based* Learning. 21-49. RIDEOUT, E. (ed.) Sudbury, Jones and Bartlett Publishers.

Schmidt, H. (1983): Problem-Based Learning: Rationale and Description. *Medical Education* **17**:11-16.

Shaw, M. and Tomayko, J. (1991): Models for Undergraduate Project Courses in Software Engineering. *CMU/SEI-91-TR-10 Technical Report*. SEI.

Sommerville, I (2001). *Software Engineering*. 6th edition. Boston, Addison Wesley.

Thomas, R. (1999): Group Dynamics and Software Engineering. *Addendum to the Proceedings of OOPSLA'99*. ACM Press.

Wells, D. (2003): *Extreme Programming: A Gentle Introduction*. http://www.extremeprogramming.org/. Accessed 26 August 2004.

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