Evaluating student teams developing unique industry projects

Nicole Clark
School of Computing
University of Tasmania
Private Bag 100, Tasmania, 7001, Australia
nclark@utas.edu.au

Abstract

Real industry projects and teamwork can have a great impact on student learning and commitment, but provision of these features also produces significant challenges for academic assessment. This paper describes an approach to evaluate teams and individuals who develop unique industry projects. The paper documents several years of experimentation with different approaches, including both failures and refinements to successful approaches. The final approach adopted allows application of the same criteria to all students even though they are working on different projects. It structures individual accountability to ensure all members of a team contribute. Assessment data is collected from multiple assessors linking them to the learning outcomes and providing the students with an understanding of personal strengths and weaknesses.

Keywords: Software Engineering, Education, Assessment, Teamwork, Industry Projects

1 Introduction

Software engineering is an integrated discipline; it is important for students to experience the practical aspects of the discipline as well as the theoretical ones to transfer successfully to industry. Many universities offer practical experience to students via a capstone course. As well as providing the obvious generic attributes of knowledge and problem-solving skills, capstone courses allow the students to develop substantial communication skills and a perspective on their industry.

Software Engineering Project is a full year capstone course undertaken by third year Computing students at the University of Tasmania. The aim of the program is to provide students with experience developing a medium-sized software engineering project in a small team of four or five students. The teams undertake unique real projects provided by local businesses to give the students industry experience. On completion, the students have an in-depth knowledge of their project domain and they have the ability to apply that knowledge in practice.

Real industry projects and teamwork can enhance student learning and commitment, but cause complications for academic assessment and as a result many lecturers avoid providing a valuable learning experience to students. Hayes, Lethbridge, Port (2003) concluded that a good grading scheme must take into account a range of information, not just rely on the final product. The challenge is to come up with a system to grade the students such that the same grading criteria can be applied to all students even though they are working on different projects. Gates, Delgado, Mondragon (2000) identified the importance of structuring individual accountability to ensure that all members of a team contribute to the project. The scheme must include an approach for measuring the individual effort of each team member. McGourty et al (2000) recommended multi-source assessment as it provides critical information from several sources, such as peers, self and lecturers, on student competencies and specific behaviours and skills, affording the student a better understanding of personal strengths and areas in need of development. They also identified the importance of linking the different assessors to the intended learning outcomes. Box (2003) recommended the use of rigorous and structured formative assessment to engage students in deep learning, encourage continuous study and increase a student's responsibility for learning.

The evaluation process for Software Engineering Project allows application of the same criteria to all students even though they are working on unique projects. The grade is determined by six main components: analysis/design, software, documentation, presentation/demonstration, professionalism, and personal software process. Components are assessed using a variety of methods and people, including self and peer assessment.

This paper begins with a description of the Software Engineering Project program. The next section presents the learning benefits provided by unique industry projects and describes the processes used to acquire the projects from industry and allocate them to the teams. The paper concludes with a discussion of the assessable components and ways to collect and analyse assessment data from multiple sources to assess teams and individuals working on unique projects.

Year	Enrolments	Teams	Feedback Responses
2002	108	24	108
2003	136	33	130
2004	130	27	98

Table 1: Software Engineering Project student data

2 Description of the program

Software Engineering Project provides students with the experience of collaborating with a real client to produce a major piece of software while working in a team and dealing with the associated problems of communication and team management. Software Engineering Project is a 26-week program offered on two campuses, broken into two 13-week units, completed in consecutive semesters; the students get two results. The team works on the same project for the entire program. The students use a software development lifecycle based on the waterfall model with testing after each phase.

Formative feedback and assessment is a major feature of Software Engineering Project. The entire software development lifecycle is repeated each semester, and a number of assessed activities are repeated within a semester. Team members, clients, testing team and the lecturer, provide formative feedback. In the first semester teams complete release one (or a third of the project), in second semester they complete release two (the remaining two-thirds). Teams receive a good grounding in the process and spend time learning the development tools in first semester. The benefit of giving them a second iteration is that they gain confidence in their abilities and familiarity with the process. They have greater knowledge of the domain and are able to tackle more technically challenging aspects of the project.

3 Why and how have unique projects?

Capstone courses in the Information, Communication, and Technology (ICT) field are offered at numerous universities using many different approaches. Fincher, Petre, Clark (2001) and Clear, Young, Goldweber, Leidig, Scott (2001) offer significant resources to capstone course lecturers to help them decide which approach to use. Projects in capstone courses can be offered in a variety of styles:

- Projects can be artificial, defined by lecturer;
- Projects can be real, defined by industry client;
- Students work on a project as individuals or in teams;
- Students or teams work on a different solution to the same real or artificial project;
- Students or teams work on a different component of the same real or artificial project;
- Students or teams work on different real or artificial projects.

Fincher, Petre, Clark (2001) provides an excellent description of the pros/cons of each approach. Over the years there have been numerous papers written that include reasons for incorporating industry projects into the curriculum: Newman (2003), Simpson (2003), and Parker (1999) are just a few. Some reasons for allowing students to experience the full software development lifecycle with a real project are:

- Students develop ability to apply skills to new contexts;
- Students experience an in-depth application of the fundamental ICT practices they have learnt and therefore internalise the concepts;
- Incorporating industry projects each year ensures that the technical skills developed by the students are relevant to the industry;
- Students are exposed to problems they will have during employment promoting deeper learning;
- Demonstrates to students that following a software development process produces quality software;
- Students develop improved professional verbal and technological communication skills with people in a variety of roles.

Some reasons for allowing the students to work in teams on a ***unique*** real industry project are:

- Allowing students to work on a project of their choice increases their motivation;
- Allowing students the experience of successfully developing software that someone will actually use increases confidence in their abilities;
- Allows new and different topics to be explored in some depths by students without changing the curriculum, e.g. different programming languages, different technology such as PDAs;
- Allows the advanced students to explore in-depth advanced ICT areas without impacting on the other students, e.g. VR systems, mobile phone software;
- Students develop more advanced problem-solving ability and critical-thinking skills because they are more committed to being successful and as a result perform at a level above their other subjects;
- With each team having their own client more students learn how to interact with and manage a real customer and thus develop a level of responsibility to others and an improved professional attitude;
- Since the team is totally responsible for the success or failure of the project they are more dedicated and develop a sense of pride that far exceeds the emotion experienced at the end of any other subject;
- Lecturers keep up-to-date with modern technology and its adoption by local industry;
- Clients get a low cost solution at minimal risk which allows the universities to improve public relations;
- Future employers find graduates have an increased understanding of business practices.

> It's the first time in the course of my degree that I've really had to deal with the real world kind of situations that are presented to you, and to be honest I enjoyed it. The sense of responsibility and feeling like what you're doing is actually relevant, or even important, to someone is fantastic.
>
> Ivan Bindoff, Nessie Team 2004

> Employers were very interested to hear about experience gained on the Project since it closely resembled the model used by many organizations today - small, tight-knit teams, working on complex problems. The group dynamics and time management skills were of particular interest to employers as these aspects are something not typically covered by a University course. When I finally found myself working in the IT Industry, I found many of the issues encountered in the Project course were likewise faced in the work place. Previous experience dealing with these issues prepared me for dealing with these on an engagement. Without having previously encountered these issues in the safety of academia, I think it would have been quite easy to make some big and very expensive mistakes.
>
> Jason Morman, Sealce Team 2000
> Sent via email in 2002

In the first two years of their degree Bachelor of Computing students at the University of Tasmania learn a great deal of theory about software engineering but practical experience is limited to artificial assignments. Software Engineering Project aims to allow students to apply this theoretical knowledge on a real-world problem so that they will be able to transfer to industry easily. The projects could be in one or more of the following domains: stand-alone applications, virtual reality systems, online content systems, systems administration software, mobile device software or artificially intelligent systems.

Each year the lecturer contacts industry members alerting them that projects are required, these people contact other interested people. The lecturer interviews each person who shows interest to see if they have a suitable project and if they will fulfil the responsibilities of a client. For reasons similar to those described by Clifton (1991) a project is suitable if:

- it is the type of project the students would experience in the industry;
- there is sufficient depth that it will take a team of students 26 weeks to develop;
- each student will have the opportunity to work both cooperatively and independently on the project; and
- students will need to undertake significant personal learning to develop increased technical skills.

The program attracts a new list of projects from industry for the students to choose from each year. The enthusiasm of industry for the program is evident by the number of industry projects on offer. In 1999 there were only 12 different projects, which was enough for the 8 teams to have a choice. Interest from the community has increased significantly over the years. Since 2002 there have been over 50 projects for the teams to choose from each year. "Word of mouth" is proving a good source of projects as past clients will talk to colleagues about the program. There is eagerness to be involved in the program, potential clients are in contact from as early as April about putting forward a project for the following year. Many clients return year after year; one ICT company has had 3 projects developed and has employed 7 students from their project teams over the years.

The projects come from a wide range of sources - a significant number come from local ICT businesses. Many businesses put forward projects to give students experience with the particular tools and languages, with an eye to employing students that take on the project. Other major sources are non-ICT businesses that simply cannot afford to have specialised software developed.

Potential clients write a short description of their project. The students must select their projects based on these short descriptions; they cannot talk to a client before allocation, as it is not practical to have over a 100 students talking to each client. If a client wants to retain the intellectual property it must be indicated in the short description. Before the students can undertake the project, they have to sign an agreement stating that the intellectual property remains with the client.

All teams are formed and projects allocated on day one during a 6-hour workshop/lecture. The students can choose their own teams – a lot of guidance is given on how to choose appropriate team members (Clark 2002). One approach is to form teams with people interested in the same project domain. Teams then choose which project they would like to do from the list of suitable projects. Teams tend to choose a project based on the ICT area they want employment in or in an area that they want to learn more about. Some students choose a project with a client simply to give them an advantage when applying for employment with them later. It gives them an opportunity to learn more about the business and to learn the tools and languages used by that client's employees.

Each team has to rank their top five project choices. It is common for more than one team to want to do the same project. The teams that clash have to give a short presentation (less than 5 minutes) 'tendering' for their project and the class vote for which team shall get the project. The unsuccessful teams move to their next ranked project, and they may have to 'tender' for that if they have competition. In 2004 for the 19 teams on one campus this process took just over an hour. In 2004 students were asked whether tendering for a project was better than the allocation being decided by the toss of a coin, 83% agreed.

Each project has a development team (that the students form themselves), but it also has a different testing team (formed by the lecturer). The testing team are students from different development teams, so students get the experience of working with people that they did not choose and the opportunity to understand and be involved with another industry project. These testing teams are required to examine all the work products produced by the development team (documents and software) before they are submitted for assessment. The formative feedback is given to the development team with time for them to consider any suggestions before they submit the work for assessment. Students learn extensively from each other, develop increased communication skills and develop a community spirit (Clark 2004).

Learning outcomes	Graduate Skills	Assessment tasks
Working in a team, students will be able to question a client to extract and analyse the software requirements and present the analysis in a written report.	Communication Skills Problem-solving Skills Knowledge Industry Perspective	Analysis Reports (2 assessments)
Having analysed the requirements, students will be able to prepare appropriate design documents while working in a team.	Communication Skills Problem-solving Skills Knowledge Industry Perspective	Design Reports (2 assessments)
Having prepared design documents, students will be able to construct and integrate a significant software system while working in a team.	Problem-solving Skills Knowledge Industry Perspective	Developed Software (2 assessments)
Having developed a software system, students will be able to construct promotional material as a team.	Communication Skills	Webpage (4 assessments) Poster/Slogan
Having developed a software system, students will be able to produce written technical and instructive documentation on the implemented solution.	Communication Skills	User manuals (2 assessments) System manual
Working as a team, students will be able to orally present and demonstrate the software system to staff and industry representatives.	Communication Skills	45 minute Presentation 4 hour Demonstration
Given reports or software developed by another team, students will be able to evaluate in writing the quality of the product.	Knowledge Problem-solving skills	Testing Reports (9 assessments)
Students will be able to formulate a schedule for a team of people and individually & collectively manage their time.	Problem-solving skills Industry Perspective	Work Plans (2 assessments) Diary (8 assessments) Timesheets (7 assessments)
Students will be able to work in a small team with a client, acting professionally and planning effectively and be able to evaluate their own and peers performance at team and individual activities.	Communication skills Problem-solving skills Industry Perspective	Self/Peer Assessment (7 assessments) Diary (8 assessments) Client meetings (5 assessments)

Table 2: Mapping of learning outcomes to assessment tasks for Software Engineering Project, 2004

4 What are the assessable components?

Hagan (2004) identified that employers want/need graduate students with skills in verbal and technological communication, problem-solving, critical-thinking, teamwork, conflict negotiation, managerial skills, and time management. Hayes, Lethbridge, Port (2003), Lister and Leaney (2003), Biggs (1999), and Box (2004) emphasised the importance of linking the learning outcomes to the assessment tasks and having an assessment scheme that reflects achievement in the graduate skills of the course. Table 2 illustrates how the learning outcomes and graduate skills relate to the assessment tasks in Software Engineering Project.

Hayes, et al (2003) also stated that the challenge is to have a system to grade the students such that application of the same grading criteria to all students even though they are working on different projects is possible. Assessment for Software Engineering Project is a 100% internal; there is no exam. There are six assessment components: analysis/design, documentation, software, professionalism, presentation/demonstration, and PSP (personal software process). Even though the program focuses on producing quality software a student's result is not based totally on successful implementation. Importance is placed on the students' ability to interact with the client, the ability to present and demonstrate their work, produce appropriate documentation and ability to work in a team. Table 3 shows the weight each component is given in semester 1 and it also shows the assessment tasks for each component. The major differences in semester 2 are that there is a combined analysis/design report worth 15%, and software is worth 40% to equate to its increased importance.

The uniqueness of the project only impacts on the quality and quantity of software produced and the type of analysis and design documentation produced. Each team does a selection of analysis and design documents depending on the type of project chosen: Requirement Document, Release Schedule, Requirement Trace Matrix, Scenarios, Storyboards, UML Diagrams, and/or Prototype Reports. The lecturer assists teams in choosing appropriate documents and as a result can ensure that all teams are undertaking a sufficient number or that teams hoping to get high grades are undertaking more work.

There are a number of assessment components that are not affected by the uniqueness of the projects. Software Engineering Project also focuses on giving the students opportunities to communicate effectively. In semester 1 the teams have to give a 45-minute presentation in front

Components	Weight	Team	Ind	Who Assess	Amount	Assessed Tasks	Worth
Design	30	20	10	Lecturer Peer/Lecturer (ind)	20 10	Analysis Report Design Report	15 15
Software	30	22	8	Lecturer Staff (2 people) Client Peer/Lecturer (ind)	7 8 7 8	Software	30
Documentation	10	6	4	Lecturer	10	Webpage User Manual	5 5
Professionalism	10	5	5	Client Lecturer Peer (ind)	3 2 5	Client Meetings Diary Teamwork	3 2 5
Presentation	10	7	3	Client Staff (2 people) Lecturer (ind)	3 4 3	Presentation	10
PSP	10	0	10	Lecturer	10	Testing Timesheets	5 5

Table 3: Assessment components for Software Engineering Project in semester 1, 2004

of staff, clients and fellow students. In semester 2 the students have to give a public demonstration of their final product. The demonstration is typically given to ICT industry members, clients, university staff, future students and the media. The demonstration days are also useful in attracting projects from industry for the following year. Teams also have to produce substantial documentation at both a user and technical level.

Software engineers have to be professional. Students have to develop skills in meeting management (such as communication and record keeping), teamwork, time management and evaluating (testing) work products produced at all phases of the software development lifecycle. In Software Engineering Project teamwork and meeting management are assessed as part of professionalism; testing and time management are assessed as part of PSP; neither of these components are affected by the uniqueness of the projects.

Each year students provide feedback on various aspects of the program, including the assessment process, by filling in an anonymous survey. The response rate is very high, see Table 1. Student feedback in 2002 resulted in the weights of each component being changed to those in table 3. Many students felt that the previous weights did not accurately reflect the amount of time spent on each component (this was also reflected in the timesheets that the students were required to complete). In 2003 and 2004 students were asked to comment on the weights of the components, the majority of the students 91% and 86% respectively, thought they were now appropriate.

Employers require graduates with teamwork experience (Hagan 2004). Unfortunately providing a student with an individual grade is difficult in a project course because the work products vary from project to project, and an individual's contribution can be hard to identify. Gates, Delgado, Mondragon (2000) identified the importance of structuring individual accountability to ensure that all members of a team contribute to the project. The scheme must include an approach for measuring the individual effort of each team member; it must identify the slackers.

In Software Engineering Project each individual receives a grade that reflects their input into the project. Each student gets an individual grade that is made up of an individual assessment worth 40% and a team assessment worth 60%; to pass a student has to get 40% of both parts and more than 50% in total. Feedback from both staff and students at the end of 2002 instigated the introduction of the 40/60 ratio, prior to that the ratio was 100% individual in semester 1 and a 100% team in semester 2. In a survey conducted at the end of 2002, 74% agreed that semester 1 was appropriate, but only 58% agreed semester 2 was appropriate. In 2003 and 2004 students were asked to comment on the 40/60 ratio and 91% and 84% respectively thought the ratio was an appropriate distribution. The most common alternative asked for by the dissenting students was a 50/50 split. Apart from the fact that the 40/60 split has the majority support it was decided to retain the emphasis on team component to encourage students to work as a team. Table 3 also indicates how much of each component forms part of the team and individual assessment.

5 Where does the assessment data come from?

McGourty, Dominick, Besterfield-Sacre, Shuman, Wolfe (2000) recommended multi-source assessment as it provides critical information from several sources, such as peers, self and lecturers, on student competencies and specific behaviours and skills, affording the student a better understanding of personal strengths and areas in need of development. They also identified the importance of linking the different assessors to the intended learning outcomes. Assessment, as well as being a measure of a student's knowledge and skills, can also be used as a learning device and that informative and useful feedback from an appropriate person is critical to achieve good learning outcomes.

In Software Engineering Project there are four main contributors of assessment data: Lecturer, Client, Staff and Students. Table 3 shows the variety of people who assess each component, and how much their assessment is worth. The rest of this section discusses how the software and professionalism components are evaluated for each student. Software assessment is tied to the uniqueness of the project, while professionalism is not tied to the uniqueness of the project. Both are good examples of involving multiple people in the assessment to provide significant feedback to students on personal strengths and weaknesses and of linking the different assessors to the intended learning outcomes.

5.1 Lecturer Evaluation

The lecturer is the person in overall control of the program. The students and lecturer meet regularly throughout the project to discuss progress, receive advice on how to proceed and to receive feedback on work products already completed.

As already stated the provision of real industry projects can produce significant challenges for academic assessment. Hayes, Lethbridge, Port (2003) stated that the grading scheme must reflect the difficulty of the project, particularly if each group is allowed to select their project to ensure equitable grades between projects. Ceddia and Dick (2004) put forward an approach for estimating project size (and therefore difficulty) using modified function points. Clark (2000) investigated a similar approach using the number of Use Cases in 1998 and 1999 but since the inclusion of other project domains where Use Cases were not applicable this approach was abandoned.

Some projects are particularly difficult and what a team can achieve in a year could be considerably less than what can be achieved on an easier project. One may question why not ensure all projects are of equal difficulty but some high achieving students really want to take on a project that is challenging and that will require significant personal learning. The lecturer gives each project a difficulty rating. This rating is used for two purposes: firstly, to help students choose a project appropriate for their abilities, and secondly, to grade the different software developed to provide standardisation.

Initially the students are given a guide to what the difficulty rating will be for that project. The initial rating is based on experience with similar projects in previous years. If a project is completely different to any previous project it will get a high difficulty rating as the students will have less resources readily available to them.

At the end of each semester when the software is being graded, the students are asked to describe what made their project difficult. The students describe what new languages, new tools or new hardware (e.g. mobile phones, PDAs, network switches) they had to use and how proficient they needed to become with them. The rating also considers access to resources, the number of students in the team (4 or 5), how many components the system developed consists of, e.g. client/server/database. The most difficult project(s) for that year are given the highest rating, e.g. a C++ program involving hardware has a rating between 90-100%. Generally the ratings drop by 5% or in some cases 10% as the difficulty degrades; the lowest rating being around 55% for a simple PHP/MySQL project.

The lecturer is also responsible for assessing the quality and quantity of the software produced for all projects. Each project is given a rating for quality, ignoring how difficult it was. This assessment includes look and feel, usability, usefulness of functionality, and structure of components. Again the process is a ranking one, with the best quality products being given the highest rating, dropping down 5% or in some cases 10% as the quality degrades. Then each project is given a rating for quantity, ignoring how difficult it was and the quality of what was produced. Quantity is based on the level of functionality and quantity of code (or equivalent). Again the process is a ranking one. These three scores are averaged to give the lecturer's team mark for software.

Obviously evaluating all the projects (33 in 2003, 27 in 2004) is a time consuming process but having one person responsible for the evaluation of all projects ensures that there is some standardisation across all projects.

The lecturer is also assessing the team diary that forms part of the team mark for professionalism. This mark is mainly a participation mark, but checks are made that they are recording important decisions and that they are reviewing progress and planning for future deadlines during their meetings.

5.2 Client Evaluation

Each team must have eight face-to-face meetings with the client; with additional electronic communication. Apart from allowing the students to extract requirements and keep the client up-to-date on progress, the meetings also allow the client to provide feedback on many of the work products and to ensure that any mistakes do not carry forward into the next phase of the software development lifecycle. The client gets to see the requirement document and release schedule in week 3, prototyped interfaces in week 6, an untested but integrated version in week 11, and release 1 in week 13. A similar pattern is followed in second semester with release 2.

Since 2000 the client has been responsible for evaluating the release produced at the end of each semester and since 2002 the professionalism of the team.

Using client assessment is a good example of linking the different assessors to the intended learning outcomes. For example, there is little point having the lecturer assess how the teams interact with their client if they are not present at the meetings, or when they are present the students radically change their behaviour. The students will learn more by having feedback from the client on their performance.

In 2004 to assess the team's professionalism the client was asked to rate the following on a 6 point scale of very poor to excellent:

- How the team communicated with them between meetings;
- How the team communicated during meetings;
- How meetings were conducted;
- How prepared the team was for meetings;
- The interest that the team showed in their project;
- The level of professionalism shown by the team.

The client does a rating three times each semester using questions similar to the above but appropriate for the timing; the final team mark for the professionalism from the client is the average of the three ratings. The students receive feedback after each assessment so that they have an opportunity to improve.

In 2004 to assess the software the client was asked to rate each release on a 6 point scale of very poor to excellent the following (similar questions have been used in previous years):

- The appearance of the user interface for the intended purpose;
- The usability of the software for its intended purpose;
- The usefulness of the functionality provided for the intended purpose;
- The level of functionality provided;
- The software as whole.

The crucial feature of all client assessment is that they can answer the questions no matter what their level of ICT experience. The client assessment on average is generally higher than the lecturer's assessment; clients like to reward the students by giving full marks when in most cases they are not warranted – particularly if the client was to compare their team's performance with another team. So over the years changes have been made to the questions to encourage clients to rate the students more accurately. The software assessment average was 88% in 2002 and 86% in 2003 and 74% in 2004. In 2002 there were 8 clients who gave 100% for software, whereas in 2003 there were only 5 and in 2004 there were only 2.

5.3 Staff Evaluation

Prior to 2002 the lecturer supervised all teams in both semesters (6 teams in 2001 and 9 in 2000). Supervisors were appointed for second semester 2002 to ease the load for the lecturer while the program was expanded over two campuses and made compulsory for all students (24 teams in 2002, 33 teams in 2003). The supervisors provided managerial advice to each of the teams in second semester and also assessed the software.

At the end of 2002 some supervisors evaluated the software for one team only and gave them an abnormally high mark for software. The staff average for software was 79% with a standard deviation of 12. The lecturer average (who was assessing them all) was 70% with a standard deviation of 15.

To attempt to rectify the assessment distribution in 2003 each supervisor was asked to supervise at least two teams, rather than just one. This had little or no effect; the staff average was 78% with a standard deviation of 9. At the end of 2003, the lecturer also felt that the work involved in supporting the supervisor team was considerable, and in fact was creating more work than it was actually saving. Supervisors were removed in 2004, and the lecturer now supervises all teams (27) in both semesters.

The benefit of involving other staff who have not been involved in the development process, is that they can provide an independent evaluation and provide additional feedback to students. In 2004 other staff still assess the quality of software produced, but the number of staff involved in the assessment is significantly reduced. In semester 1 2004 one staff member assessed 17 projects, and another staff member assessed 23 projects. The plan was to use both staff members to assess them all but illness and timetables prevented it. The staff used the same software assessment questions as the clients. The average software mark in semester 1 2004 was 67% and the standard deviation was 11. The lecturer's average software mark was 66% with a standard deviation of 12.

6 Student Self/Peer Evaluation

Current educational practice is advocating allowing students to do self and peer assessment to increase student learning (Biggs 1999). Wilkins and Lawhead (2000) advise using a variety of techniques to reduce the possibility of a student intentionally damaging the score of another student.

One interesting feature about Software Engineering Project is the use of self and peer assessment to evaluate the software and professionalism components, as well as other components. A range of tools are used and Clark et al (2005) provides a detailed discussion on their evolution:

- Timesheets
- Self/Peer Evaluation Surveys
- Individual Contribution Report
- Quantitative Report

Prior to 2004, a student's ability to work in a team was assessed by the lecturer; students felt this was a problem as it meant they couldn't discuss team management issues with the lecturer (the person most likely to be able to help them) without it affecting their assessment (even though they were told many times it wouldn't). In 2004 their team members used a peer evaluation survey form, similar to that devised by Sanders (1984), to assess each student's individual professionalism; an example can be found in Clark, Davies, Skeers (2005). Each question is worth 5 marks and the result for a survey is the average. The students complete the forms four times a semester; sometimes only for formative feedback. The questions are divided into two groups: Meetings and Work Habits. The form asks about such things as attendance at team meetings, contributing to the ideas and discussion at the meetings, completing work by deadlines set at team

meetings, and their ability to work with team members. For each question the student indicates how often the team member behaved according to the scale: Always, Usually, Sometimes, Rarely, Never. By allowing the students to do the assessment, it is linking the assessors to the intended learning outcomes. The students will learn more by having feedback from their peers on their performance.

Each student's level of professionalism is discussed at the team meetings with the lecturer, so guidance on how to improve can be given. In the survey conducted at the end of semester 1 2004 students were asked if the peer survey forms were a good way to give feedback on team members performance, 88% agreed.

As stated by McGourty, Dominick, Reilly (1998) "*In a cooperative learning environment, students themselves are often in the best position to provide one another with meaningful feedback regarding both their technical and interpersonal performance*". In 2004 students use an online version of the peer evaluation survey form and a student is able to see the aggregate results of their team member's feedback (preserving individual anonymity), and hence have a chance to improve before the next assessment. In the 2004 survey students were asked if the peer feedback resulted in a change in their performance: of the 98 students who responded to the survey, 2 people never looked at the feedback form, 27 looked at it, but it had no effect, 62 people felt they made slight changes and 7 people said they changed a lot.

To help assess each individual's contribution to a major work product (such as software) students are required to write a report reflecting on their contribution. This is similar to the process described by many others (e.g. Gates et al (2000), Hayes et al (2003)). In the review of 2003, 85% of students felt the individual contribution reports were a fair way of indicating their contributions. In 2004 an online peer evaluation system allows the other team members to read the reports and indicate agreement or disagreement (with explanation), which has overcome one of the previous grievances that students felt their team members may be lying in their contribution reports. Similar to the approach described by Clear (2002) the reports are discussed at the team meetings with the lecturer when a significant disagreement arises. In 2004 the students were again asked whether the individual contribution reports were a fair way of indicating their contributions, 90% agreed.

Another tool used to help assess an individual's contribution is to have each team member give a quantitative opinion of how much each team member contributed to a major work product. This approach is similar to that described by Hayes, Lethbridge, Port (2003) and is used similar to the way Brown (1995) used the Autorating system. In semester 2 2004 the students had $100 (virtual) to distribute between team members based on the quality and quantity of work produced by each individual for each major work product (such as software). If a student believed everyone contributed equally then they would give equal amounts, or if they believed that someone did more valuable work, they would give that person more, and others less. Each student could also give a student a bonus of $5 if they believed an individual had done above and beyond what was required (or asked for). They could only give one team member a bonus, and they could not give it to themselves. They had to provide an explanation of why they were giving a bonus.

Hayes, Lethbridge, Port (2003) reported that the University of Southern California uses a similar approach but that they have the students do the forms weekly. In Software Engineering Project the students are required to do them with each work product submission to allow students to have weeks where project is not their main focus. Students find it hard to hide a lack of contribution from their peers, particularly around submission time.

As with Brown's (1995) Autorating system the forms have relieved the lecturer of the difficult task of accurately assessing individual student contribution in teams. Both Brown (1995) and Hayes et al (2003) expressed some concerns about groups ganging up on a member or covering for an idle individual and recommended that the quantity tool is not used in isolation. In Software Engineering Project the lecturer confirms the numbers by examining some of the work produced by individuals, the individual contribution reports and the timesheets for each student and if the numbers do not accurately reflect the contribution of an individual they are adjusted. This process is particularly important where there is wide variation in allocations by team members, or where the individual allocates themselves significantly more than their team members.

The assessment data for the software component from the four contributors are combined to give each student a result for software (SW) using the following process:

Step 1: Evaluate team mark for the software:
SW(team) = SW(lecturer)+SW(staff)+SW(client)

Step 2: Examine individual values (pay) to determine if they correlate with the individual contribution reports, timesheets, and work submitted. Adjust if necessary.

Step 3: Convert pay to a percentage (ind%)

Step 4: SW(ind) = SW(team)*ind%* Teamsize*8
note: 8 is the individual weight for software

Step 5: SW = SW(ind) + SW(team)

The software (SW) is worth up to 30% of the maximum mark in semester 1. Step 1 will give each team a software mark out of 22, SW(team), and is simply the sum of each assessor's mark. Step 3 will give each team member an individual contribution percentage, ind%, which when added together should total 100%, if there were no bonuses. Step 4 gives an individual mark out of 8, SW(ind), that can be higher than 8 if a person does more than their share, eg 30% in a 4 person team. If the team mark is assessed at less than full marks, say 20 out of 22 for software, and an individual has 10 for the individual mark, they would get 30/30 for software. The individual excess is counted to reward team members who are doing more or better work than their team members.

7 Discussion

At first glance the amount of data and the number of sources of data may make the approach seem complex. In practice, it does not prove so. It simply requires a systematic approach to gather all the data from the different sources. The students move from one phase of the lifecycle to the next and data for each assessment component can be collected independently. It is necessary to impose strict deadlines on everyone involved.

The approach may also appear demanding for the lecturer. The approach was devised for a lecturer who has to manage (supervise) a large number of teams. To reduce the workload for that lecturer, mechanisms to automate the collection of data were developed. The approach can be adapted to share assessment of different components if you are lucky enough to have a team of people involved. It is advised that the same person do all the assessment of that component for every team, e.g. the same person assess all the user manuals, and another could do all the web pages. It is recommended that the lecturer (or person meeting regularly with each team) does the standardisation assessment of the major work products that involve individual assessment, such as software, as the meetings give insights into how the team is operating.

8 Conclusion

There are many reasons for incorporating real unique industry projects: increased student motivation and confidence; students can explore in-depth ICT areas not covered in the curriculum or only covered superficially; students develop increased problem-solving and critical-thinking skills, communication skills and business acumen.

Even though the assessment is challenging there is no reason for lecturers to shy away from unique projects. Structure the assessment process such that application of the same criteria to all projects is possible and the uniqueness of the project only affects a few of the criteria.

If the students are also allowed to work in teams the scheme must structure individual accountability and sourcing data from a variety of people is recommended as it gives the students a better understanding of their strengths and weaknesses and it is possible to better relate the assessment data to the learning outcomes.

9 Acknowledgements

I would like to acknowledge the contribution of the students, clients and supervisors who have been involved with projects from 1999 to 2004. I would also like to acknowledge Ms Pam Davies as she developed the online versions of the peer evaluation forms in 2004.

10 References

BIGGS, J. (1999): *Teaching for Quality Learning at University*. Buckingham, England, Society for Research into Higher Education and Open University.

BOX, I. (2003): Assessing the Assessment: an Empirical Study of an Information Systems Development Subject. *Proceedings of the fifth Australasian Conference on Computing Education*. Adelaide, Australia, **20**:149-158

BOX, I. (2004): Object-Oriented Analysis, Criterion-Referencing and Bloom. *Proceedings of the sixth Australasian Conference on Computing Education*. Dunedin, NZ, **30**:1 – 8

BROWN, R. (1995): Autorating: Getting Individual Marks from Team Marks and Enhancing Teamwork. *Proceedings of Frontiers in Education Conference*.

CEDDIA, J., DICK, M. (2004): Automating the Estimation of Project Size from Software Design Tools Using Modified Function Points. *Proceedings of the sixth Australasian Conference on Computing Education*. Dunedin, NZ, **30**:33-39.

CLARK, N. (2000): Pilot Projects for Object-Oriented Design: An Empirical Study. *Proceedings of the 12th Australian Software Engineering Conference*, Canberra, 139-148

CLARK, N. (2002): Software Engineering Projects: Working in Teams. *Proceedings of the sixth IASTED International Conference: Software Engineering and Applications*, Cambridge USA, 698-703

CLARK, N. (2004): Peer Testing in Software Engineering Projects. *Proceedings of the sixth Australasian Conference on Computing Education*. Dunedin, NZ, **30**:41-48

CLARK, N., DAVIES, P., SKEERS, R. (2005): Self and Peer Assessment in Software Engineering Project. *Proceedings of the seventh Australasian Conference on Computing Education*. Newcastle, Australia

CLEAR, T., YOUNG, F.H., GOLDWEBER, M., LEIDIG, P.M., SCOTT, K. (2001): Resources for Instructors of Capstone Courses in Computing. *Proceedings of Annual Joint Conference Integrating Technology into Computer Science Education*, Canterbury UK, **33**(4):93-113

CLEAR, T. (2002): A Diagnostic Technique for Addressing Group Performance in Capstone Projects. *Proceedings of Annual Joint Conference Integrating Technology into Computer Science Education*, Aarhus, Denmark, 196

CLIFTON, J.M. (1991): An Industry Approach to the Software Engineering Course. *Proceedings of the 22nd SIGCSE Technical Symposium on Computer Science Education*, San Antonio, Texas, **23**(1):296-299

FINCHER, S., PETRE, M., CLARK, M. (2001): *Computer Science Project Work Principles and Pragmatics*. Springer-Verlag, London

GATES, A.Q., DELGADO, N., MONDRAGON, O. (2000): A Structured Approach for Managing a Practical Software Engineering Course. *ASEE/IEEE Frontiers in Education Conference*, October, Kansas City. **1**:21-26

HAGAN, D. (2004): Employer Satisfaction with ICT graduates. *Proceedings of the sixth Australasian Conference on Computing Education*., NZ, **30**:119-124

HAYES, J.H., LETHBRIDGE, T.C., PORT, D. (2003): Evaluating Individual Contribution Toward Group Software Engineering Projects. *Proceedings of International Conference on Software Engineering, Portland*, Oregon. 622-627

LISTER, R., LEANEY, J. (2003): Introductory programming, criterion-referencing, and Bloom. *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, Nevada, **20**:143-147

MCGOURTY, J., DOMINICK, P., REILLY, R. (1998): Incorporating Student Peer Review and Feedback into the Assessment Process. *Proceedings of Frontiers in Education Conference*, Tempe, Arizona, 14-18

MCGOURTY, J., DOMINICK, P., BESTERFIELD-SACRE, M., SHUMAN, L., WOLFE, H. (2000): Improving Student Learning through the use of Multisource Assessment and Feedback. *ASEE/IEEE Frontiers in Education Conference*, Kansas City. **1**:21-26

NEWMAN, I., DANIELS, M., FAULKNER, X. (2003): Open Ended Group Projects a Tool for More Effective Teaching. *Proceedings of the fifth Australasian Conference on Computing Education*. Adelaide, Australia, **20**:95-103

PARKER, H., HOLCOMBE, M. (1999): Campus-based Industrial Software Projects: risks and rewards. *Proceedings of Annual Joint Conference Integrating Technology into Computer Science Education*, Cracow Poland, **31**(3):189

SANDERS, D. (1984): Managing and Evaluating Students in a Directed Project Course. *SIGCSE Bulletin,* **16**(1): 15-18

SIMPSON, M., BURMEISTER, J., BOYKIW, A., ZHU, J. (2003): Successful Studio-based Real-World Projects in IT Education. *Proceedings of the fifth Australasian Conference on Computing Education*. Adelaide, Australia, **20**:41-51

WILKINS, D.E., LAWHEAD, P.B. (2000): Evaluating Individuals in Team Projects. *Proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education,* Austin, Texas, 172-175