

Assessing Competency in Undergraduate Software Engineering Teams

Marie Devlin

School of Computing Science
Newcastle University
Newcastle, UK

Chris Phillips

School of Computing Science
Newcastle University
Newcastle, UK

Abstract - From 2005, Active Learning in Computing partners Newcastle and Durham University (ALiC), part of the UK CETL initiative,[1], introduced a collaborative learning model of Software Engineering to level 2 Computing Science students that reflects global industry practice by focusing on cross-site software development. Assessment for this effort focuses on measuring students' development of both the technical and transferable skills associated with the practice of being a software engineer. However, it is often difficult for the student to perceive and articulate what skills they have learned during the project based on marks and feedback from lots of separate elements of coursework. In this paper, we propose that assessment of Software Engineering team projects should focus on the development of a range of competencies that could be measured in a style that relates directly to professional performance appraisal. We describe the current assessment methods we use and then outline a set of alternative competencies and appraisal methods that could be used to help staff and students better evaluate levels of achievement and skill development in qualitative terms during undergraduate team projects in Software Engineering.

Keywords: Software Engineering, Competency, Skill development

I. INTRODUCTION

The Centre of Excellence in Teaching and Learning, (CETL), Active Learning in Computing, (ALiC), which commenced in 2005, is a consortium of North East UK universities comprising of Durham University (CETL lead), Newcastle University, Leeds Metropolitan University and the University of Leeds. The initiative aims to identify and enable ways in which students can become more actively engaged with the Computing Science curriculum in their learning. Through promoting project and group-working, ALiC has implemented new learning approaches, enabling students to move towards independent learning guided by appropriate support materials. To that end, ALiC partners Newcastle and Durham have introduced a collaborative learning model of Software Engineering to level 2 Computing Science students that reflects global industry practice by focusing on cross-site software development. Industrial software is often produced collaboratively between teams located at different geographical sites. The pedagogical aims of this cross-site collaboration activity are therefore: to give students an insight into Software Engineering in an industrial context; make

problem-solving more realistic in student team projects; allow staff and students to use and evaluate various technologies for cooperative working and encourage the development of transferable skills such as communication, organising and team-working. Skills outcomes for the module at Newcastle were and still are listed as: initiative; adaptability; teamwork; numeracy; problem-solving; interpersonal communication and oral presentation. Overall, students have reported good learning outcomes in questionnaires and focus groups set up to evaluate the module design and their experiences. However, assigning marks to students during and at the end of the module on many separate pieces of coursework [2] means that student achievement is "abstracted into just a few numbers" so it is often difficult for students to perceive and articulate what software engineering skills they have learned and how these skills have developed during the project.

In this paper we describe the learning design of the cross-site development project. We review the assessment methods currently used to evaluate student performance and measure student learning outcomes. We propose that assessment of undergraduate team projects in Software Engineering should focus on the development of a range of competencies similar to those identified by Turley and Bieman, in conjunction with the assessment of technical and team work products in a style that relates to professional performance appraisal [3]. This approach would give students a richer view of their achievements and a more useful and realistic performance review of how their competency has changed and developed throughout the project. We outline a set of alternative competencies and appraisal methods that could be used to provide staff and students with a better understanding of levels of achievement and skill development during undergraduate team projects in Software Engineering.

II. LEARNING DESIGN OF THE MODULE

At the beginning of the academic year, teams are formed from the Software Engineering module cohort at Newcastle and each one is then paired with a corresponding team at Durham. The major project task is the design and implementation of a large software system – (e.g. in 2005 the task was a tour guide application that could be loaded onto a PDA or mobile phone, in 2006 they had to create a logistics system for UK-wide distribution of personal care products and in 2007, teams had

to develop a virtual geo-caching application). The aims of the module are to introduce the real world experience of team working and to provide practical experience of large scale software development. The teams of students work must work together as a ‘virtual’ enterprise across the sites for the whole academic year, using communication technologies to facilitate their collaboration.

There are some differences as to how Software Engineering is taught at Newcastle and Durham and through four iterations of the cross-site development project we have altered our practices to accommodate these differences and make the module a more cohesive experience for students in terms of our approach [5]. In our learning design for Software Engineering at Newcastle we ‘front-load’ the lecture component of the module and give our students 10 one hour introductory lectures on the basics of Software Engineering including an overview lifecycle models and phases and their associated techniques and tools. In operational terms, Newcastle take a problem-based learning approach – where students must rely on their existing programming knowledge and direct their own learning to solve the problem after the introductory lectures. Durham have adopted a hybrid-approach of traditional Computer Science teaching mixed with partial problem-based learning. They allow students to make all the team decisions but provide lectures and supervised laboratory practical sessions for the whole year to support the student development effort. The decisions the virtual companies are responsible include the definition and allocation of roles; the outline and implementation of a company management structure; the allocation of tasks to each member; the definition of the project plan; the choice of development methodology; tools to be used to deliver their software and documentation at the end.

Other deliverables for the project include coursework elements that are to be completed on an individual or local team basis e.g. a personal skills assessment, a contract, as well as product demonstrations. Each iteration of the cross-site model has seen minor changes to the nature of deliverables depending on the size of the cohort and also the nature of the problem to be solved. We have also evolved the way we teach the module each time it has run, making changes based on student and employer feedback.

III. CURRENT ASSESSMENT METHODS

One element of the project that remains unchangeable is the need for each individual institution to take responsibility for the assessment of their students. It is likely that any institution’s policy would dictate the need for this in a cross-site development project in order to ensure that institutional standards and integrity are maintained. Separate assessment protects students from the possible consequences of working with poorly performing students at another institution. In practice, this has meant that staff at both institutions work closely to ensure that students are fully aware of the aims and motivation of the assignment as early as possible and to ensure that a mark for any work product reflects their institution’s students’ contribution. Good assessment is fair, valid and

reliable. We believe that it should relate to real tasks and realistic contexts, make expectations and criteria clear and provide feedback that allows a student to progress in their learning. It should also accurately measure skills and qualities that are practiced, demonstrated and improved throughout the student’s learning experience.

There are a lot of assessments for the project throughout the year, (approximately 17). Before the CETL altered the learning design, there were 34 assessments each year. Table 1 illustrates the current set of assessments on the module and their type i.e. whether they are individual or team efforts. The rationale for having so many was to ensure that students had as many opportunities as possible to demonstrate and practice their skills and improve on their learning. Having many assignments also ensured that teams were kept busy and had enough work to distribute amongst their members. As part of the changes we made for the cross-site work, we also removed some of these assessments as some skills were over-assessed during the module in previous years e.g. presentation skills were assessed three times in pre-CETL iterations.

TABLE I. ASSESSMENT TYPES

<i>Assessment</i>	<i>Type</i>
Strengths Essay	Individual
Review	Team
Interim Requirements (F) *	Team
Group Structure	Individual
Contract	Team
Interim Design (F) *	Team
Presentation	Team
Project Plan	Team
Testing Strategy	Team
Final Requirements	Team
Final Report	Team & Individual
User Manual	Team
Final Design	Team
Software Submission	Team
Peer Assessments	Individual
Software Demo	Team
Monitor Observations	Individual

a. (F) – Formative Assessment – feedback only

The students are given formal templates for both the Requirements Specification and Design documents and allowed to submit a copy of these for formative feedback (as indicated by (F) in Table I before submitting their final efforts for grading. Students find this reassuring and it means they can make mistakes and improve their performance during the

module, rather than retrospectively as in most forms of academic assessment.

There is a lot of emphasis in the module on documentation and on assessing team processes as well as products. At Newcastle team processes are *monitored* once a week by members of staff who observe each team's formal meeting. These observations of interaction and participation are given a grade by the monitor at the end of the project and act as a weighting for the overall team mark. One reason for the emphasis on documentation are to help students to realise the importance of industry-standard notations such as UML in describing design. Having documentation that is assessed also ensures those students who are weaker in terms of coding proficiency can contribute in valuable ways to the whole team effort and be rewarded.

Table I also illustrates that there is a lot of joint assessment between sites. With each iteration of the cross-site work we increased the dependency between the teams for grades and the level of joint marking. The reason for increasing this dependency between teams was to increase their motivation to collaborate properly. However, the downside to the increase in dependency between sites is that it has also increased the level of anxiety students have about team assessment. This means that staff at both institutions have had to work hard on specifying common marking criteria that suits the aims of their separate programs and is also explicit in determining how an individual grade will be derived from all the cross-site and local team efforts.

A. Defining Individual Contribution

Students are often nervous or anxious about assessment fairness when involved in a group effort and cross-site working further exacerbated these fears. So, in order to ensure the student contribution at each site was differentiated clearly we mandated that students use a simple contribution matrix to help ensure recognition of individual effort. The matrix illustrates the type and amount of effort each individual contributed to an element of assessment, regardless of their base location. An example of a design document is given in Figure 1. In this matrix, the sections of the design document are outlined and then each student's contribution is defined in terms of whether they created the section, modified it or reviewed it in some way. The contribution matrices take a lot of effort to fill in for students especially for larger pieces of work such as coding but this ensures they are specific about the nature of the work they have done and the pieces of work they have contributed to.

Sections	Joe	Kirill	Michael	Tom
1.0 Introduction	C	M	R	M
1.1 Purpose	CMR	R	CM	R
2.1.1 PC Modules	Durham	Durham	Durham	Durham
2.1.2 PDA Modules	M	M	C	
3.1.1 PC Modules	CMR	CMR	R	
3.1.2 PDA Modules	Durham	Durham	Durham	

c=create
m=modify
r=review

Figure 1. sample contribution matrix

The use of these matrices helped to alleviate some of the assessment fears faced by students about working in a group e.g. the impact of non-participation of some members or the fear of not getting enough credit for their efforts. However, even though we were very specific about marking criteria and students reported that the matrices helped, they were still anxious about assessment so we decided to review all our assessment methods.

B. Coders and Non-Coders

We examined student grades from the Software Engineering module at Newcastle for 2 years before and 2 years after the implementation of cross-site work in order to determine how the changes we made had impacted on student performance. Because of the differences between cohorts each year and the subtle changes we made in each iteration to deal with operational problems, we could only infer this impact and note general trends. We analysed student grades, team reports and individual reflective reports from the years 2003-2007.

We found that, in general, students who did not contribute largely to the coding of the product during the project, received lower grades, on average, across all the years.

As can be seen in Table II there were 109 coders (34% of the total number of completing students) and 215 coders. The data indicated that 57% of non-coders scored less than their team's average mark and 39% of coders. These figures were quite worrying considering that the coding effort and subsequent software produced is worth only 5% of the total module marks available. The reason that the product is worth so very little in terms of grade percentage is because we want to emphasise all the other aspects of Software Engineering that are equally as important as the end product. Students tend to focus very much on the product and view programming as the only real important part of the work they are doing. As tutors we need them to recognise that Software Engineering is more than just programming, therefore we place more emphasis and give more academic credit to other deliverables and processes.

TABLE II. MARKS OF CODERS AND NON-CODERS FROM NEWCASTLE

Implement	1 st	2.1	2.2	3rd
No	26	42	31	9
Yes	67	91	46	8
Total	93	133	77	17

The intended learning outcomes for the module are divided into knowledge and skills outcomes as illustrated in Table III. These learning outcomes are sufficiently broad to cater for a range of abilities amongst the student cohort and terms such as *practical experience in design and implementation* (see Table III: Skill Outcomes column), should cater for all the processes associated with the design and implementation of a system, including the non-coding aspects. If a student has demonstrated these learning outcomes (to varying levels), then the differences between coders and non-coders should be irrelevant. The results from our analysis of marks made us wonder are coders better Software Engineers or even just

better students? Ideally, for us, assessment in the module should be based on performance and not innate ability (although this is a factor that cannot be ignored) – however it is all too easy to specify a set of criteria for performance based on our perceptions of ability rather than what students actually do.

IV. MEASURING LEARNING OUTCOMES

The worries over assessment of contributions the differences in marks between coders and non-coders set us thinking about what we really are assessing and what the student gets out of the module in terms of learning outcomes and skills. We had lots of questions that needed an answer e.g. does the mark generated by the assessment of all the deliverables and assignments give a true reflection of what the student has learned about Software Engineering? Is a mark for the project a sufficient indicator for an employer in industry of the skills a student possesses? Does the module give the student real tangible skills that can be used as soon as they go into a job?. Does it matter if they programmed the system or took some other role in the teamwork? Can students interpret what they have learned in the module as a realistic software engineering experience? Can students see how their skills have progressed and developed during the project?

TABLE III. MODULE LEARNING OUTCOMES

Intended Knowledge Outcomes	Intended Skill Outcomes
An understanding of the issues that relate to planning and execution of a team-based software project	Practical experience in the design and implementation of a large software system.
An understanding of the software engineering process, process models and stages	Practical experience in issues such as team structure, document preparation, project management.
	The ability to work as a member of a team, to fulfil appropriate roles within a team
	Evaluate own learning, progress and quality of solution objectively.
	Technical writing. Report writing
	Critical self-evaluation, peer evaluation
	Presentation of results.

As part of the module we run a series of mock interviews with real employers. Each student applies for a position, sending their CV and a covering letter. The employers taking part often complain that students do not present themselves as well as they could at interview. Some of the problems are definitely due to a lack of confidence but also many students focus only on their technical skills in the CV and at the interview and ignore the ‘soft’ skills that we also want them to learn during the project. Most employers want a more rounded individual with the potential to grow and change in the organisation, a self-starter that works well both in teams and on their own. So students need to be able to articulate what the module has taught them both in terms of technical software engineering skills and other transferable skills e.g. communication and leadership. One thing that we currently do well is get students

to complete a self-assessment skills ticklist and essay at the start of the year. We give them a set of broad categories based on an old version of Belbin team roles [10]. Students are asked to identify their existing skills and where they might fit in the common stages of the software lifecycle e.g. would they fit into a design role, are there skills they have that could be used in all phases of development?

We ask the students to complete a self-assessment report at the end of the module (Final Report, Table I). This report encourages the student to reflect on their project experience and discuss any skills they have developed, improved or learned for the first time. They are asked to compare their initial skills assessment to their skill levels at the end of the project. Over 4 iterations of the project, by reading these reports and conducting focus groups with the module cohort, it has become clear that students do not always know the reason why they are doing a particular assessment or task during the project. Some students also have had difficulty grasping the importance of formative assessment of large work products and often fail to use the feedback we give them to improve their grades. Others have difficulty in articulating how their skills have changed or what new skills they may have gained during the project.

Our experiences tell us that students have not reached a critical understanding of many of the important aspects we are trying to teach them and we feel that this is caused by the nature of the assessment tasks we set them – they are, to some extent mismatched with what we are trying to achieve. According to Ambrose “the extent to which students engage in critical reflection in large part depends on the extent to which the assessment methods employed have developed critical understanding as their goal.” [8]. We feel this is an area where our assessment design could be improved. Our assessments are too academic and bureaucratic in nature. Whilst work such as the Requirements and Design documents gives students practice using formal notations and techniques associated with Software Engineering, students receive numerical values and feedback for each single document or software item (which is collaboratively-produced) as a measurement of their proficiency and competency-levels in the discipline. This makes it difficult for them to get an overview of their personal performance as a software engineer. Individual assignments such as essays, presentations and reports don’t really test or assess their practical skills – merely their reflective skills. Practical skills are treated as a group commodity and often assigned one grade or a weighted grade but the feedback is common to all. We use methods to derive an individual’s marks for the large products, but not individual feedback. Observations made by monitors can be subjective and don’t reflect a true picture of the practical work students do outside of formal meetings and class times. The *process* is not being measured as accurately as it could be and feedback to students needs to be more meaningful on an individual level in order to capture the wealth of learning they have experienced and give them a real sense of their ability as a software engineer.

V. ASSESSING COMPETENCY

Competency matrices, such as those used by Smith and Smarkusky for self and peer assessment [6] could be used

instead to capture learning and give students rich feedback on their abilities and skill levels. A competency matrix captures team knowledge and skills in various categories (process, communication, interaction, contribution and responsibility). These matrices then allow an assessor to assign a numerical range of proficiency in each specified competency – individuals are evaluated by selecting a class rank to indicate the baseline competencies expected of the individual. In this work, peers assess whether an individual has met the expectations, exceeded the expectation by various amounts or requires improvement (varying amounts of improvement can be denoted e.g. we have used first, second or third class performance). An abbreviated example adapted from the work of Smith and Smarkusky is given in table IV.

TABLE IV. SMITH AND SMARKUSKY’S COMPETENCY MATRICES

CLASS RANK	1 ST	2 ND	3 RD
Process	Steps required to complete project		
Task Performance	Exhibits on tasks behaviour consistently	Supports others in completing tasks	Motivates others to independently stay on task
Leadership skills	Learns about leadership skills	Rehearses leadership skills	Exercises leadership skills
Communication	Oral and written means by which students share ideas and information.		
Shares ideas	Shares ideas especially when asked	Shares ideas readily	Mentors others in sharing ideas
Asks questions	Asks questions for communication	Asks questions related to project fundamentals	Asks question that directs conversation toward project solution
Interaction	Ways in which students interact socially, their interpersonal skills, how they resolve conflicts.		
Team problem-solving skills	Uses critical thinking to develop a project solution	Encourages a team decision by examining project criteria and supporting evidence of each alternative	Uses methodology to make a team decision and presents problem solution with supporting evidence from each team member.

Currently, our module learning outcomes for the Software Engineering Team Project are quite explicit in that we expect students to undertake roles, practice project management skills, communication skills etc. but in terms of Software Engineering competency – our assessment instruments are weak. Ambrose [8] suggests we need to provide an holistic view of a person’s competency and for this to happen students need to develop self-efficacy where they can make judgements not on what skills they have but what they can do with the skills they

possess. We do this in a small way at the start of our project, by getting the students to evaluate their skills in broad terms using a reduced Belbin form [10] that denotes broad team characteristics and behaviours team members might exhibit or possess innately e.g. an Investigator personality is good at finding things out, will be particular about finding new ways to solve problems etc. However, we do not retain the focus on skills in other pieces of coursework throughout the year or at the end. The focus on skills tends to get lost in the creation and assessment of the software engineering products that we ask the students to complete and the broad criteria that we use to assess them. Feedback for the module tends to focus on one piece of coursework at a time and does not give the students an overarching view of how they are performing during the project or at the end. Ambrose’s work and the work of Marakas et al [9] in the area of measuring competency suggest antecedents to self-efficacy include verbal persuasion by a credible mentor, social comparison, (by observing someone else performing similar tasks) and the degree and quality of feedback and perceived effort can all enhance or decrease self-efficacy beliefs. As outlined by Ambrose, traditional measures of programmer’s competencies include experience, professional references, training, transcripts and academic references, professional certification, written, oral and other demonstrative assessments during job interviews. These are all useful ways to find evidence of how well a professional is viewed and has performed in the field but don’t really provide us with a guide to assessing students who are novices to the discipline of software engineering and whose degree program is not totally focused on software engineering. So how can we evaluate software engineering competency in an educational setting? We believe an adapted set of competency matrices could be used as part of an overall appraisal-like process for students throughout the whole project. Students would have to evaluate themselves and the performance of others in their team. They could be provided with examples of how to assess in a workshop before the project commences to instruct them on how to approach appraisal and fill in the matrices for themselves and others.

As these judgements could be subject to bias, other forms of assessment such as summative grading of work products, the use of signed and agreed contribution matrices for tangible deliverables and an individual report and interview at the end of the process would be used. The student evaluations could be used in conjunction with those of staff. Staff would use the same matrices and criteria to assess students as the students themselves and then compare matrices in order to assign grades based on a scale defining if they have met expectations, exceeded expectations or if they need to work on certain areas. This would give a sense of continuity, a transparency in assessment and a shared language for staff and students to discuss performance. This method would also give students some experience of ‘social comparison’. Students very rarely get to view the results and feedback given to others and therefore find it hard to compare their performance to that of others in their class in a formal or managed way. A competency matrix would also give students early feedback as

to their progress so they can correct poor behaviours. We could use the matrices periodically to get students to evaluate how they think they are performing along a set of pre-defined competency areas.

VI. SOFTWARE ENGINEERING COMPETENCIES

We propose that the competencies that should be measured via peer, self, formative and summative assessment are along the lines of those discovered by Turley and Bieman when conducting a study of exceptional and non-exceptional professional software engineers [4]. They identified 38 competencies including – helps others, willingness to confront others, responds to schedule pressure, focus on user/customer needs, team-oriented, writes / automates tests with code etc. We would use these in conjunction with the assessment of technical and team work products. Many of the behaviours Turley and Bieman identified with non-exceptional performance “can be viewed as the behaviours of inexperienced engineers” because a beginner “will be unsure of their own skills and capabilities” and therefore defining levels of proficiency or development in these behaviours should give our students more confidence as to how they have improved during the year. Suggested competency areas are 24 of the 38 that were identified by Turley and Bieman, (as illustrated in Table V).

TABLE V. TURLEY AND BIEMAN’S ESSENTIAL COMPETENCIES

Team Oriented	Seeks Help	Helps Others
Use of Prototypes	Writes Tests with code	Knowledge
Obtains Necessary Training/ learning	Communication	Methodical Problem Solving
Uses Code Reading	Response to schedule pressure	Sense of mission
Attention to detail	Perserverance	Innovation
Desire to improve things	Focus on user or customer needs	Sense of Fun
Lack of Ego	Thinking	Skills/Techniques
Quality	Elegant and simple solutions	Thoroughness

The authors define competencies as “the skills, techniques and attributes of job performance” [3, 4]. They conducted an in-depth interview with 20 professional software engineers, (10 ‘non-exceptional and 10 ‘exceptional’) who were employed by a major computing firm, (using the Critical Incident Interview technique as outlined by John Flanagan in 1954, the precursor of competency modelling) [12]. They also analysed competencies identified by software managers. The competencies they identified provide an alternative way of looking at the job of software engineering. The competencies are organised into four categories – *Task Accomplishment*, *Personal Attributes*, *Situational Skills* and *Interpersonal Skills*. In Table V, for example, Task Accomplishment competencies are Methodical Problem Solving, Obtains Necessary Training/Learning and Skills/Techniques. The authors provide examples of each competency e.g. Skills/Techniques is defined as proficient in using design

techniques, debugging skills and easily makes technology choices. The authors concluded that although “most of the competencies cannot be used to distinguish between the exceptional and non-exceptional subjects, the derived competencies offer a unique view of the skills of professional software engineers” [3]. These competencies could be used to in conjunction with graded or evaluated competency matrices to help us illustrate the progression for the student on a software engineering module or even a whole software engineering program. This would mean they would get feedback on their skill development as opposed to only grades for a series of documents and work products that may not give them a complete picture of their learning or of what is expected in terms of the behaviour of a professional software engineer.

VII. THE STUDENT-APPRAISAL METHOD

We propose that an initial skills assessment is used before a team project begins in order to define the starting position, and past experience of students.

This will give students an idea of the skills they already have and how they could be best put to use during the project. At intervals during the project, (perhaps at the end of software development phases), staff could use the matrices to rate students based on observation of meetings and their performance in presenting or talking about their work. Students could use the same matrices to rate themselves and their team mates according to those competencies identified by Turley and Beiman, for example on:

- Using knowledge
- Researching, Investigating, Problem-Solving
- Communicating with the rest of their team – methods and quality
- Developing solutions
- Speaking and presenting in groups
- Technical appreciation – use of software, hardware
- Taking initiative and responsibility
- Understanding of main duties and responsibilities
- Their most important achievements of the phase of development
- Their planning and response to schedule pressures.

These matrices would help us gauge the level of confidence a student has in their abilities and could also help initiate a discussion of roles and possibilities for learning during the project. We would then use a further set of matrices during the project, to measure the new competency levels of the students and assign grades. The matrices would also help us determine target competencies for students to work on and improve during the rest of the project phases. The matrices could also help us determine the group level of competency and provide early interventions if teams are failing. This type of assessment, from self, peers and tutors (similar to an employee 360⁰ appraisal review by subordinates, peers and superiors) would give students an insight into how their

professional work will be assessed by employers and colleagues. The matrices would be used in conjunction with the traditional assessment of technical work products and tangible deliverables such as code and documentation. This assessment approach would also include an interview on experiences and skills at the end of the project. We believe that although an individual interview may add to the resource burden of teaching for team projects, the incorporation of an interview will give students a more rounded and personal review of their performance. All team project experiences are designed to give students a realistic experience of working on a large piece of software and an insight into what it is like to work on real problems within development teams. The proposed methods of assessment we outline here offer an opportunity for students to receive higher quality feedback on their progress and development as software engineers and to determine how to further develop their skills, in a safe environment – which is the point of most team project exercises in higher education at undergraduate level.

The work outlined here could be adapted quite readily for other disciplines and not just Software Engineering. The current matrices that we have created and proposed focus on skills and knowledge that has been identified in practice and in the literature as key to the work of professional engineers, however it would be feasible to identify generic skills and competencies required in team working scenarios for other disciplines. Rarely does a professional person work in isolation these days and generic skills such as communication, leadership, negotiation and problem-solving are required of most professionals in the modern workplace. These generic teamworking skills could be substituted in the matrices and used as a basis for the competency evaluation and the student appraisal method we have outlined here.

VIII. CONCLUSIONS

In this paper we have described our cross-site software development project and some of the difficulties students have with our assessment methods and with determining their progress and skill development as software engineers during the project. A lot of the problems that students face are due to our use of academic assessment methods that provide a numerical grade and feedback on separate pieces of work, some of which has been constructed by the team. We have outlined a new method that seeks to give individual students feedback on their progress during the project in a richer and more holistic manner. We have introduced a contribution matrix method to help ensure individual effort and contribution are noted and the correct grades attributed to each student. We review competency matrices and competency definitions outlined in previous work and propose a new appraisal method that uses self, peer and tutor assessment of alternative software engineering competencies and skills. We believe this new appraisal-style method will help students to get better feedback on their performance, make assessment criteria more transparent and help students recognise and articulate their development and skills as software engineers more clearly.

IX. FURTHER WORK

Some of the positive impacts of the cross-site work described here are greater students' awareness of how differences in working practices, organisational culture, team structure, task allocation and project management styles between teams can impact on project outcomes. However assessment and feedback are areas that need improvement. We are continuing to develop and refine our competency-based approach to assessment and hope to develop a wider framework for assessment of undergraduate software engineering team projects and automated tools to support them.

REFERENCES

- [1] HEFCE CETL Initiative: <http://www.hefce.ac.uk/learning/tinits/cetl/> Accessed 10/11/09
- [2] R.A. McNamara, "Evaluating Assessment with Competency Mapping", Proceedings of the Sixth Conference on Australasian Computing Education, vol. 30, R. Lister and A. Young (eds), ACM International Conference Proceeding Series, vol. 57, Australian Computer Society, Darlinghurst, Australia, pp 193-99, 2004.
- [3] R.T. Turley and J.M. Bieman, "Competencies of Exceptional and Non-exceptional Software Engineers, Journal of Systems and Software, vol. 28, issue 1, pp 19-38, 1995.
- [4] R.T. Turley and J.M. Bieman, "Identifying Essential Competencies of Software Engineers", Proceedings of the 22nd annual ACM computer science conference on Scaling up : meeting the challenge of complexity in real-world computing applications: meeting the challenge of complexity in real-world computing applications, Phoenix, Arizona, United States, pp 271 – 278, 1994
- [5] Devlin, M., Drummond, S., Phillips, C. and Marshall, L. "Improving Assessment in Software Engineering Student Team Projects" In 9th Annual Conference of the Subject Centre for Information and Computer Sciences, 26th-28th August 2008, Liverpool Hope University White, H. (ed.), Higher Education Academy, Subject Centre for ICS, pp 133-139, 2008.
- [6] Harold H Smith, 111, Debera L Smarkusky. "Competency Matrices for Peer Assessment of Individuals in Team Projects" in Proceedings of SIGITE '05, pp155-161,2005.
- [7] K. Wilson, J. Fowler, "Assessing the impact of learning environments on students' approaches to learning: comparing conventional and action learning designs", Assessment and Evaluation of Higher Education, vol. 30, No.1, pp87-101, 2005.
- [8] Paul J Ambrose, Issues in Information Systems, vol. VIII, No. 2, pp273-279, 2007
- [9] Sanjay Goel, "Competency Focused Engineering Education with reference to IT related disciplines: Is the Indian System Ready for Transformation?", Journal of Information Technology Education, vol. 5, pp27-52, 2006.
- [10] Gibbs, G., *Learning in Teams: A Student Manual*, (1981), Oxford Centre for Staff Development, 1994, ISBN 1 873576 20 X, which is in turn based on *Management Teams*, R.M. Belbin, Heinemann, 1981.
- [11] Marakas, G. Yi, M.Y., & Johnson, R.D., "The Multilevel and Multifaceted Character of Computer Self-Efficacy: Toward Clarification of the Construct and an Integrative Framework for Research", Information Systems Research, 9, (2), pp126-163, 1998.
- [12] Flanagan, J.C. (1954). 'The critical incident technique', *Psychological Bulletin*, vol. 51, pp327-58

