Science and Mathematics Education Centre

Evaluating the Effectiveness of a Real-Life, Team Based Software Development Project for Tertiary Students

Sandra Cleland

This thesis is presented for the Degree of

Masters of Philosophy

of

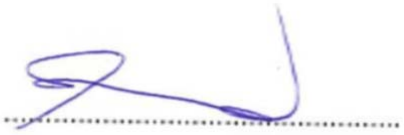
Curtin University

November 2013

Declaration

To the best of my knowledge and belief this thesis contains no material previously published by any other person except where due acknowledgment has been made.

This thesis contains no material which has been accepted for the award of any other degree or diploma in any university.

Signature: 

Abstract

The purpose of this action research study was to evaluate the effectiveness of a real-life, team-based, software development project at preparing Information and Communications Technology (ICT) students for junior developer roles. Empirical evidence suggests that graduate developers are ill-prepared for the realities of their first position. Traditionally, most software development courses involve project work where students develop a software application from scratch based on stable user requirements. This does little to prepare them for the type of software maintenance work they are likely to be given as graduates entering the workforce and does not expose them to the realities of working with real users who often have conflicting and changing requirements. Bachelor of Information and Communications Technology students undertaking their 300 level Software Engineering course are involved in a real-life project where they have to work as a team in order to better prepare them for the real world. This study involved: surveying of graduates of the Software Engineering course to determine the extent of skills developed within this learning environment, to identify how useful the skills have been, and to find common gaps in their skill sets experienced as new employees; adaptation of the learning environment to address the identified skill gaps; surveying a selection of regional employers to identify core skills required of their ICT graduates; observation of students working within the adapted environment, and analysis of their reflective journals. The results of the graduate surveys show that a majority of the skills developed within the learning environment are useful to the graduates once in the workforce. Confidence in their abilities as developers improved, and a majority of them used the experience as evidence during the job seeking process. Survey responses from the employer sample confirmed the importance of a majority of the skills developed within the learning environment; in particular the soft skills were seen as important by all employer respondents. Observations of students utilising the adapted environment and their reflective journals were analysed to identify problem areas and devise strategies to alleviate issues in future iterations of the project. Results show that real-life, team-based development projects do indeed help to bridge the gap between the academic world and industry practice.

Acknowledgements

My appreciation goes out to my patient family and the graduates of the software engineering course who shared their valuable time and thoughts with me.

Table of Contents

List of Tables

List of Figures

CHAPTER ONE

1.1 Introduction

It has long been recognised that traditional software engineering education has been lacking in teaching students skills that will make them ready for the challenges of the workplace. Research suggests that graduate developers are ill-prepared for the realities of their first position (Begel & Simon, 2008a, 2008b; Borenstein, 1992; Brechner, 2003; Dawson, 2000; Dawson, Newsham, & Fernley, 1997; Denning, 1992; Lethbridge, 1998b). Traditionally, software development courses often involve project work where students, in small teams, develop a software product (from the initial design to the final coded application), based on stable user requirements. This learning environment does little to prepare students for the type of work they are likely to be given as junior developers, that of software maintenance and bug-fixing.

In 1983, a company called GPT (now known as Siemens GEC Communication Systems Ltd) started an in-house training course for their computer science graduates with the aim of highlighting "the differences of working in the real world compared with the near 'ideal' environment experienced at university" (Dawson et al., 1997, p. 287). The course aimed to simulate the real-world by: demonstrating to the new hires that customers often have conflicting requirements; showing them that the requirements are likely to change during the project; making it clear that project disasters are not exceptional; and helping them develop the personal skills of communication, planning, and adaptability required to work effectively in a team (Dawson, 2000).

Similarly, an observational study undertaken by Begel and Simon (2008b) on graduate software developers at Microsoft Corporation discussed that while new graduates were proficient in: coding, reading and writing of design specifications, and problem solving; they experienced difficulties in the following areas: communication, collaboration, technical tools used for large-scale development, cognition, and orientation within their project teams where there may be little in the way of well organised information.

Research suggests that the way to help prepare software development students for the realities of their first job is to involve them in a project where: there is a real client or role play of a real client; the client requirements change or clients have conflicting priorities; students are made to work in teams; and the team works on an application with a large existing codebase (Begel & Simon, 2008a; Coppit, 2006; Dawson, 2000; Dawson et al., 1997; Joy, 2005; Upchurch & Sims-Knight, 1998).

Whilst the real-life project approach has been implemented in many software engineering courses, at the time of data collection the researcher found no evidence in the literature that the students who partake in this approach have been asked to evaluate the relevance of

their experience once they are in the workforce. This research aimed to fill this gap by asking graduate developers in the workforce for feedback on the real-life, team-based development learning environment they participated in; leading to a re-design of the learning environment to include more relevant tools and techniques.

1.2 Context

Within the New Zealand education system, polytechnics are government funded technical institutes that focus on vocational training. The hands-on, practical nature of the courses offered within these institutions make them ideally placed to offer training in Information and Communications Technology (ICT). The polytechnics applied approach means their ICT graduates are sought after by employers as they enter the workforce with practical skills making them effective employees from their first day on the job. A majority of New Zealand polytechnics offer ICT undergraduate degrees as well as certificate and diploma level qualifications. Currently, the New Zealand Qualifications Framework lists 13 polytechnics as offering a Bachelor degree in ICT (NZQA, 2012).

The researcher's institute, UCOL, is located in Palmerston North, New Zealand and is one of the larger 14 regional polytechnics. UCOL provides training for Software Engineers within the Bachelor of Information and Communications Technology (Applied). This degree contains four core software development courses. Students who are interested in becoming Software Engineers take all four core courses, and also develop complementary skill sets by taking other database, analysis, and web development options within the degree. The core development courses begin with a compulsory level 5 Software Development course where students learn programming fundamentals (control structures, graphical user interface design, and use of an integrated development environment). Interested students then select an optional level 6 course, Advanced Programming, which focuses on teaching more advanced topics such as: threading, database access using data sources, and LINQ (Language Integrated Querying). Another optional level 6 development course is also available, Software Process, which begins training developers in software management processes and exposes the students to their first experience in team-based development. The main aims of the level 7 course, Software Engineering, are:

- Construct software of professional standard in a repeatable manner
- Secure applications ready for distribution
- Conduct effective software inspections

The researcher is the lecturer for this course. The researcher implemented a real-life, team-based software development project as the learning environment for the level 7 Software Engineering course. The research participants consisted of: past students of the Software Engineering course, now working as junior developers; a selection of regional ICT

employers; and current students who took the course utilising the adapted and improved learning environment.

1.3 Aim and Research Questions

This action research study aimed to reflect on the different software tools and team-based development skills used within the third year Software Engineering learning environment and evaluated the relevance of the skills learnt in relation to what was required of the graduates in their first development role and the identified skill sets required by regional employers.

The study aimed to answer the following research questions:

1. Do non-technical skills (e.g. communication, problem solving and decision making, self-management, and teamwork) continue to be of high importance to employers hiring ICT graduates?

2. Are the development languages, technologies, and processes utilised in the project still currently in demand in local industry?

3. What skills did the graduates perceive as improved or formed by participating in the real-life development project?

4. Of the skills demonstrated during the project which ones were required of the graduate developers in their first ICT role?

5. Did the learning environment provide real-world experience that the novice developers could use as evidence of skills during employment interviews?

6. Are there common themes amongst the skills the graduates identify as lacking in their knowledge base that could be addressed by adaptation of the learning environment?

1.4 Research Methods

The action research cycle began with the construction of a learning environment within the software engineering course that aimed to address the inadequacies of traditional software developer education as described in previous research (Begel & Simon, 2008a; Coppit, 2006; Dawson, 2000; Dawson et al., 1997; Joy, 2005; Upchurch & Sims-Knight, 1998; Borenstein, 1992; Brechner, 2002; Lethbridge, 1998b). For the purpose of this study the learning environment is defined as: the physical classroom environment within which the participants would interact, including specific seating locations designed to improve team interaction; the virtual environment comprised of team based development tools and tools used for team communication; and the psychological environment involving real client interaction (or role-play of the client) and team interaction.

After the initial run of the software engineering course with the newly constructed learning environment, feedback from the graduate participants and their employers was used to redesign the learning environment to address identified skill gaps for the second iteration of the real-life project. Iteration two graduates and their employers were then surveyed and their feedback led to another redesign of the learning environment. In the third iteration of the project, classroom observations and participant reflective journals were used to identify difficulties experienced by the students working within the learning environment allowing the researcher to devise strategies to help students overcome these hurdles in future iterations of the real-life project.

The action research study used a convenience sample obtained from students and graduates of the researcher's third year Software Engineering class who participated in the real-life, team-based software development project and who volunteered to be part of the study (8 graduates, 20 students). A convenience sample of regional ICT employers was obtained from employers of the graduates being surveyed and organisations known to the researcher through the Bachelor of Information and Commmunications Technology Industry Advisory Committee (8 employers). Graduates who had been employed fulltime in software developer roles prior to studying were not included in the sample as they would not be classed as novice developers.

Preliminary Activities

A literature review was undertaken. This included an examination of the previous software engineering class that was involved in the real-life team-based project to identify the participants and document the learning environment activities and software resources used.

Data Collection Phase One

Graduates, who gained employment in developer roles, were surveyed in order to determine the skills developed within this learning environment and to try and identify common gaps in their skill sets experienced as new employees. For each of the different skills demonstrated within the learning environment, graduates were asked to rate how much they learnt of that skill in the project and how useful it had been to them so far in their career. This is an adoption of a scale applied by Lethbridge (1998b) in his survey of software professionals to gauge their perception of the relevance of their education.

The employers of the graduates were surveyed to identify the core skills required of their new ICT graduates. Questions for this survey were created using categories adapted from previous local studies (Bekesi & Gardner, 2003; Snell, Snell-Siddle, & Whitehouse, 2002; Young, Senadheera, & Clear, 1999) and skill surveys conducted by international recruitment agencies and IT consultants (Argent, 2006; Watson, 2010).

Phase One Data Analysis and Learning Environment Adaption

Qualitative analysis of the questionnaires yielded data for:

- evaluating the usefulness of the learning environment at preparing the graduates for work
- helping identify common gaps in skill sets

This was followed by an investigation into adaptation of the learning environment to address the identified skill gaps, followed by implementation of viable changes.

Data Collection Phase Two

Graduates, from the second iteration of the real-life, team-based project who gained employment in developer roles, were surveyed to review the effectiveness of the adapted environment. This survey aimed to validate the relevance of the skills developed within the adapted learning environment and to identify if there were still gaps in their skill sets after the redesign. The employers of these graduates were also surveyed to identify the core skills required of their new ICT graduates.

Phase Two Data Analysis and Learning Environment Adaption

Qualitative analysis of the questionnaires yielded data for:

- evaluating the usefulness of the adapted learning environment at preparing the graduates for work
- helping identify common gaps in skill sets

This was followed by an investigation into adaptation of the learning environment to address the skill gaps, followed by implementation of viable changes.

Data Collection Phase Three

The final stage of the research project involved classroom observations of students working within the adapted environment. Students who participated in this iteration of the real-life team-based project were required to complete reflective journals documenting their development experience. These journals were used, alongside the classroom observations to gain an insight into major difficulties experienced by the students working within the adapted environment to allow the researcher to aid students to overcome these hurdles in future iterations of the real-life, team-based project.

1.5 Significance

Though there are a number of researchers who have identified inadequacies in traditional software engineering education and suggested strategies for providing students with a more realistic learning environment, there appears to be have been little evaluation of how the real-life project experiences and the skills gained from them have been viewed by graduates once they are in the workforce.

This research aims to validate and improve the learning environment provided to the software engineering students on the Bachelor of Information and Communications Technology (Applied) degree with the aim of making graduate transition to the workplace more seamless.

It will ensure the skills being taught within the learning environment make the graduates more employable and address the requirements of the regional ICT community.

The information on the learning environment and techniques used to help students overcome difficulties faced in the classroom will add to the body of knowledge on tertiary teaching methods, particularly in relation to technology based courses.

The research on the effectiveness of the learning environment may be useful to other institutions in New Zealand looking to implement a similar approach.

1.6 Limitations

Due to small class sizes for two of the three years that this research was conducted the survey sample of graduates is small (eight responses from a possible eleven graduates). This also led to a small employer sample (eight responses).

1.7 Overview of Thesis

Empirical evidence suggests that graduate developers are ill-prepared for the realities of their first position. Traditionally, most software development courses involve project work where students develop a software application from scratch based on stable user requirements. This does little to prepare them for the type of software maintenance work they are likely to be given as graduates entering the workforce. To better prepare students for transition into the workforce it is generally agreed that software development students should: work as a team using team-based development tools, be exposed to an application with a large existing code base, have a real-life client or role play of such, have a project with changing requirements or be made to prioritise conflicting requirements.

The researcher implemented a real-life, team-based software development project as the learning environment for the third year Software Engineering course taught within the Bachelor of Information and Communications Technology (Applied) at UCOL, Palmerston North, New Zealand. Though many organisations have implemented this approach, there appears to be little evidence in the literature that the graduates of such a learning environment have been asked to provide feedback on the experience once they have entered the workforce.

The researcher surveyed graduates of the learning environment to yield data to evaluate the usefulness of the learning environment at preparing the graduates for work and identify common gaps in skill sets. This feedback led to modification of the learning environment.

The graduates of the second iteration of the project were surveyed to gain feedback on the usefulness of the adapted learning environment and to determine if there were still gaps remaining in their skill sets. The employers of the graduates were surveyed to identify the core skills required of their new ICT graduates to confirm the relevance of adapted learning environment. The final iteration of the research project involved classroom observations of a larger cohort of students working within the adapted environment. The observations were used alongside the students' reflective journals to identify challenges experienced working within the adapted environment to allow the researcher to aid students to overcome these hurdles in future iterations of the real-life, team-based project.

The chapter that follows contains the literature review undertaken for the research project. The literature review is broken into six key sections: the debate over whether software development is an engineering practice or a science discipline, the traditional approach to education and the new demands of the knowledge society, an overview of the history of software developer education, industry perspectives on the skill requirements of junior developers, modern techniques used within courses to try and meet the demands of industry, and real-world team-based development projects that are trying to bridge the gap between theory and work readiness. Chapter Three describes the research methods adopted for the project. The research project covered three iterations of the learning environment to allow feedback to be obtained on the adaptions made. The learning environment for each phase of the data collection is described and the methods used to capture the data are detailed. Chapter Four describes the results of the data collection for the three iterations of the learning environment. In the concluding chapter the research objectives are re-examined against the results, the limitations and significance of the research are restated and suggestions for future research are made.

CHAPTER TWO

2.1 Introduction

It has long been recognised that traditional software engineering education has been lacking in teaching software development students skills that will make them ready for the challenges of the workplace (Begel & Simon, 2008a, 2008b; Borenstein, 1992; Brechner, 2003; Lethbridge, 1998b). Since the dawn of the computing era, educators have argued over the best way to help their learners develop the necessary thinking skills required to work within the ICT industry. Should graduates be schooled in traditional scientific methods to learn their trade, or should engineering practices be imparted? Even though this has been a long, rigorous debate and both types of teaching approaches have been implemented in multiple tertiary institutions worldwide, empirical evidence suggests that graduate developers are ill-prepared for the realities of their first position (Borenstein, 1992; Dawson, 2000; Dawson et al., 1997; Lethbridge, 2000). Traditionally, most software development courses involve project work where students develop a software application from scratch based on stable user requirements. This does not prepare them for the type of software maintenance work they are likely to be given as graduates entering the workforce, or arm them with the knowledge that user requirements are often in conflict and subject to change.

This literature review begins by investigating the debate that what is actually required of junior developers is an engineering discipline, not a computer science background. This is followed by a discussion of the traditional approach to education, the demands of the new knowledge society, and the educational techniques historically used to train software developers. Then the skills required by industry of junior software developers are outlined and the Software Engineers Body of Knowledge is discussed. Following this the techniques and tools that have been introduced by some educational institutions to modernise courses and align with industry developments are described. Finally, the concept of real-world, team-based projects bridging the gap between software engineering theory and workplace practice is discussed.

2.2 It's Software Engineering, not Computer Science

"The scientist builds in order to study; the engineer studies in order to build"
(Brooks, 1996, p. 62)

There has been debate among the academic community for decades regarding the fundamental nature of computing education: is it a science discipline or an engineering practice? The discussion has its foundations in the birth of the computer science departments and the first international Software Engineering conference held in 1968 (Bauer, 1973; Brooks, 1996; McConnell, 1998). Bauer (1973) describes the use of the term 'Software Engineering' by the NATO Scientific Committee in the conference title as being

provocative, it was used to try to highlight the fact that there was “something lacking in the computer field” (p. 476).

Frederick P. Brooks (1996) first wrote of the misnomer of the naming of the University of North Carolina’s newly formed ‘Computer Science’ department in 1977. On receiving the ACM Alan Newell Award in 1994 he provided lifetime reflections on his discipline in an acceptance speech where he reiterated his ideas of how rather than a science, ICT is an engineering discipline (Brooks, 1996). Brooks (1996) described science as concentrating on the “discovery of facts and laws” and that the discovery of any new fact or new law “is an accomplishment, worthy of publication”; in contrast, engineers “are concerned with making things, be they computers, algorithms, or software systems” (p. 62). Unlike other engineering disciplines, much of what is produced is intangible; software engineers do not make things that “directly satisfy human needs”, the things they create are used by others “in making things that enrich human living” (Brooks, 1996, p. 62). Brooks (1996) states the computer scientist is a tool smith whose achievements are demonstrated through successful use of the tools they create, therefore an engineering education is more useful than a scientific background.

Boehm (1976) first defined Software Engineering as “the practical application of scientific knowledge in the design and construction of computer programs and the associated documentation required to develop, operate, and maintain them” (p. 1226). He describes the practice of Software Engineering as involving the following elements: requirements engineering, preliminary design, detailed design, programming, testing, and maintenance (Boehm, 1976). Boehm (1976) provided an assessment of the techniques used in software development at that time that were based on “solid scientific principles (versus empirical heuristics)” (p. 1239). There was little in the way of solid scientific principles for software development in comparison with what was available for hardware development.

His assessment covered four different scopes: application of scientific principles across the entire software development life cycle; principles spanning the entire range of software applications; those concerned with engineering economic analysis; and those relating to the personnel developing the applications (Boehm, 1976). Scientific principles across the life cycle saw a few applicable to component design and development (e.g., algorithms, automata theory), but very little relating to systems design and integration; across applications there were some principles relating to systems software (e.g., discrete mathematical structures) but none for application software; there were a few principles applicable to the system economics (e.g., algorithms); and a few principles able to be applied by the technicians coding the software (e.g., basic maths packages, structured code) (Boehm, 1976). Boehm (1976) concluded that at that time there was little in the way of scientific principles to underpin the software engineering practices of requirements analysis, design, test, and maintenance.

McConnell (1998) believes that software development should be an engineering practice as "engineering is the application of scientific principles toward practical ends" (p. 118). He points out similarities between construction projects and software projects, in both cases the engineer is required to select the appropriate materials to suit the product's purpose with any wastage in resources frowned upon (McConnell, 1998). Opposers of this stance state no core body of knowledge exists that describes 'Software Engineering', so therefore it cannot be described as an engineering discipline (McConnell, 1998). In response to this, McConnell (1998) describes the building up of a core body of knowledge over the decades since the first conference on Software Engineering was held in 1968. This body of knowledge now includes practices in: requirements elicitation, design, coding, system integration, cost and time estimations, and quality assurance (McConnell, 1998).

Newell and Simon (1976) describe Computer Science as empirical enquiry, stating each new machine that is built, and each new program that is written are experiments that can be observed and analysed to discover new phenomena, or help with further understanding of phenomena already discovered. They assert that Computer Science is indeed a science in that its practitioners "develop scientific hypotheses which [they] then seek to verify by empirical inquiry" (Newell & Simon, 1976, p. 120).

Parnas (1999) offers a clear distinction between the type of knowledge acquired through a scientific or an engineering education. A scientist's main purpose is to add to the body of knowledge for their discipline. Therefore, they must learn: "what is true" – the existing body of knowledge of the area of interest; how to test the hypotheses; and how to extend the body of knowledge for their field (Parnas, 1999, p. 22). The main purpose of an engineer is to build reliable and quality products fit for purpose. Therefore, engineers must learn: "what is true and useful in their speciality" – the existing body of knowledge; how to apply it; and how to apply knowledge from a broader domain to solve practical problems (Parnas, 1999, p. 22).

Parnas (1999) states that both types of approaches are necessary as the graduates of each will have quite different career paths. The engineering path will be taken by those who see themselves as builders, designers of tools for others; the scientific path followed by those who wish to investigate things of interest to both groups to add to the body of knowledge (Parnas, 1999).

Bauer (1973) states "Software Engineering means to obtain a quality controlled product under economical considerations" (p. 476). He believes the computer science students "should be prepared for real life, not a scientific life that is not the real life for 95% of them" (Bauer, 1973, p. 479). Bauer (1973) does not, however, believe that this takes an entirely new curriculum suggesting that engineering aspects can be added to computer science courses through additional subjects and teaching approaches. "Scientific education has a number of advantages: it removes the stupid and exercises the brains of those who remain, giving them great intellectual confidence and the feeling that they can cope with problems"

(Bauer, 1973, p. 478). Bauer (1973) describes the viewpoint that engineering "requires experience and experience is something one goes through rather than something that is taught" (p. 478).

It is clear that to be a skilled software developer one requires knowledge of and experience in engineering practices to ensure building of quality software tools, fit for purpose and able to be maintained. With both types of degree having been implemented in tertiary institutions worldwide, computer science with a long history and the newly designed software engineering programs, the empirical evidence still suggests that graduate developers are ill-prepared for the realities of their first position (Begel & Simon, 2008a, 2008b; Brechner, 2003; Dawson, 2000; Dawson et al., 1997; Lethbridge, 1998b). This stems in part from the traditional model of education and, in particular, the traditional methods of educating software developers.

2.3 Traditional education and the demands of the knowledge society

In his 1992 article, *Educating a new engineer*, Peter Denning describes an education system that has evolved from being the responsibility of local community and church to a state-managed and run process, with class sizes and the number of institutions growing exponentially in the last 50 years (Denning, 1992). State-funded research began during the Second World War when the US government offered contracts to some Universities to have staff and students focus their attention towards answering questions that would aid the war effort; this funding became legislated practice with the creation of the National Science Foundation in 1945 (Denning, 1992). Scientists were encouraged to research areas that would benefit society in terms of military security, health, and economic security with international competitiveness, the manufacturing initiative, and the high performance computing and communications program being added as federal objectives in the 1980s (Denning, 1992). These government incentives led universities to restructure their reward systems into what we have today; success became delivering those research outputs that targeted the government sanctioned research activities that attracted funding.

Denning (1992) comments that this approach, commonly known as the "publish or perish" reward system, has led to faculty members desperate for research outputs and reluctant to spend time in the classroom; this has led to students complaining that the faculty value research more than them and that they are unable to obtain good career advice from their professors as they have contact mainly with the junior teaching assistants (p. 84). The 'publish or perish' system is common amongst universities in the western world. Denning (1992) sees the research focused university model and the traditional approaches to computer science education as doing little to prepare "graduates for the world in which they will actually work"; continuing on to say that students are lacking in "the areas of communication and collaboration, rather than technologies" (p. 88). Denning (1992)

describes the need to adapt curriculum and the traditional approach to education so that graduates are competent in skills needed by the modern organisation.

Drucker (1988) writes of the modern organisation, the information-based organisation, where knowledge workers are specialists working in task-focused teams. Once advanced technology use propagates throughout an organisation, the organisational structure tends to flatten as middle management, who were there mainly to act as relays of information, are no longer required (Drucker, 1988). The knowledge within the modern organisation lies at the bottom, in the minds of the specialist, self-directed workers that are formed into teams (Drucker, 1988). Denning (1992) describes the computing professional as an "expert partner", a person who works with clients in other domains to provide them with information system support (p. 86).

This need for the IT professional to become an expert partner who supports others within the organisation became even more imperative as society transitioned from having a few information-based organisations into what became known as the Information Society, where most organisations are heavily reliant on information systems. Anderson (2008) states that by the 1980s the term 'Information Society' was used to describe the "explosion of information and information systems" (p. 5). The Information Society metaphor was weakened in the 1990s with the emergence of the Knowledge Society, a reference to economic systems where products are ideas or knowledge (Anderson, 2008).

To support these shifts in the way work is completed within the modern organisation, educational curriculums needed more emphasis on communication and team-based work. Denning (1992) proposes some changes that need to be made to traditional education so graduates are prepared for the world in which they will work: students should learn skills by applying them under the guidance of someone who is already proficient in the skill; students should be regularly made to adapt and learn something new; faculty should act as guides and coaches not just presenters of information; students must learn to work effectively in groups; subject material should be "balanced between informational knowledge – facts, procedures, models, and processes - and action knowledge – which can be learned only through involvement with other people" (p. 89). Anderson (2008) outlines the new demands placed on graduates in the knowledge society in Table 2.1.

Table 2.1: Implications of the demands of the global knowledge economy for youth in terms of required skills and learning strategies (Anderson, 2008, p. 7)

Demands from society	Required skills	Learning strategies
Knowledge as commodity	Knowledge construction	Inquiry, project learning, constructivism
Rapid change, renewal	Adaptability	Learning to relearn, on-demand learning
Information explosion	Finding, organizing, retrieving information, ICT usage	Multi-database browsing exercises
Poorly organised information	Information management, ICT utilization	Database design and implementation
Incompletely evaluated information	Critical thinking	Evaluation problem solving
Collectivization of knowledge	Teamwork	Collaborative learning

These skills, identified as being necessary for anyone entering the workforce, are even more critical for graduates in ICT disciplines where rapid change and complexity are the norm. The traditional approaches to education of software developers needed to adapt even more so than other disciplines.

2.3.1 Traditional education of software developers – a history

Bauer (1973) discusses the evolution of programming and programming education. He describes three stages of progression in the early decades of the computing era: the classical period; the pioneering age; and the software crisis (Bauer, 1973). The classical period, from 1945 – 1955, saw the arrival of the first fully mechanical, programmable computer and was dominated by the engineers who built these early machines; therefore the fascination was in the technical capabilities of the machine (particularly the electronics) and not the programming of it (Bauer, 1973). This led to neglect of the software side of things and resulted in no training of programmers in this period; programming was something the specialist (e.g., engineer, physicist, mathematician) had to do for themselves (Bauer, 1973).

The next period, the pioneering age from 1955 -1965, saw the commercial use of computers begin which resulted in an explosion of sales of the machinery and an increasing “demand for personnel to be delivered with the computer”, refer Figure 2.1 (Bauer, 1973, p. 472).

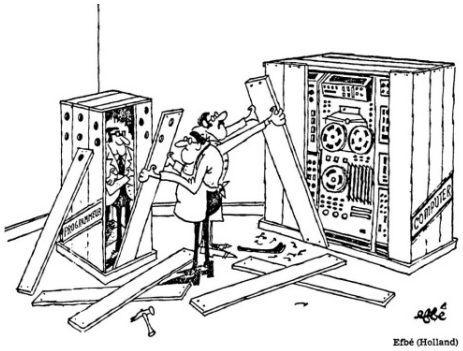


Figure 2.1: Personnel delivered with the computer

The manufacturers coped with the demand by cajoling as many intelligent beings as they could into the field, leading to programmers with little or no training and “an unwillingness to catch up” (Bauer, 1973). The need for formal training was being felt and many universities responded with Computing Science or Computer Science, the choice of term was under dispute as was the content of the courses (Bauer, 1973). Bauer (1973) states that during this time some of the problems faced by educational institutes were to do with establishing a Ph.D in the field of Computer Science; this also influenced peripheral subject choices as did the location of the department within the university itself. For example: the areas of automata theory and artificial intelligence were frequently enhanced under the Computer Science banner as it was both easier to obtain a professor and gain a Ph.D in these fields than in system programming; Stanford University when it set up its Computer Science department included numerical analysis because it did not become part of the Mathematics department (Bauer, 1973). Bauer (1973) quotes Perlis to sum up the state of programming education at this point: “What is good for computing science is what is good for writing a Ph.D thesis in computing science” (p. 474). So despite the attempts of educational institutes to start to address the need for programming training the pioneering age concluded with still very few experienced and able system programmers (Bauer, 1973).

In the next decade, the computing era entered into the period known as the “Software Crisis” which was characterised by computer systems that were being built bigger and bigger, as there were no costly raw materials, but definitely not better (Bauer, 1973). The increasingly large systems consumed large amounts of expensive hardware storage and suffered performance problems, they were also unreliable as they were full of flaws (Bauer, 1973). Hamming (cited in Bauer, 1973) summarised the issues of the software crisis: systems were not delivered on time and they often didn’t run; systems that did run were full of bugs; programmers were poor communicators who were unable to deliver on time and co-operate with others.

Bauer (1973) points out that during this period the way in which manufacturers set up maintenance arrangements for their software could only lead to the conclusion that the users paying for the product were expected to field test the software and find the bugs. The manufacturers would only address problems with the software when the client demanded help and often the fix would introduce new bugs, which would again be ignored until the client demanded it be fixed (Bauer, 1973). This type of behaviour led Dijkstra (cited in Bauer, 1973), a participant at the NATO software engineering conference in 1968, to state “the whole business is based on one big fraud” (p. 476). Considering software releases from some of the major software vendors today, not much has changed in that respect. A beta testing phase has been introduced where the companies release a product for worldwide user testing. However, this only addresses major flaws and the users still end up paying for software which will undergo regular updating to address bugs found by users over time.

During the software crisis period there were major improvements in the techniques used for software development, such as the use of control structures and decreased use of the goto statement as advocated by Dijkstra; techniques that came to be known as structured programming (Dijkstra, 1968, 1972). These new practices started to flow through to some of the Computer Science curriculums. Bauer (1973) provides an example of an advanced course delivered in Munich in 1972 comprised of topics such as: structured design, language hierarchies, generic components, efficiency and reliability, verification and quality control. Bauer (1973) closes his summary of programming history with a look at the educational problem faced in the decade that would follow, 1975-85. He expresses the need for "fundamental changes in the habits of the programming community" that can only be bought about by education; he suggests that there should be a supervised practical work component to the education of software developers so they can practice their skills while being mentored (p. 477). He concludes by stating that future graduates need to make software development a profession by using engineering practices with a scientific viewpoint (Bauer, 1973).

The computing industry was growing rapidly and educators were struggling to keep pace with the changes in this new field. As previously discussed in section 2.2, there was an on-going debate around whether computer science degrees were the best place to be educating software developers, or should engineering departments be teaching software developers engineering principles and practices. The computer science courses were being criticised as not meeting the needs of industry. Borenstein (1992) states that the skills focused in on by these courses are unrelated to the tasks graduate programmers will be engaged in once employed.

In 1976, the first suggested framework for a separate software engineering curriculum was proposed by Freeman, Wasserman, and Fairley (1976). Freeman et al. (1976) describe the software engineer as a generalist who takes on a multitude of roles: "problem solver, designer, implementor [sic], manager, facilitator, and communicator" (p. 116). To provide the skills needed by these generalists who worked in projects where there is often time and budget constraints, as well as unclear user requirements, five content areas were suggested to underpin any future training programs in software engineering; the areas to be covered were: computer science, management science, communication skills, problem solving, and design (Freeman et al., 1976).

Computer science would cover knowledge in: algorithms, common software structures, hardware, programming methodologies and languages, testing and debugging; Management science had suggested topics of: cost estimation, user requirements analysis, budgeting, organisational theory, human resource management, and industrial psychology; Communication skills would address: written communication, oral communication, interpersonal communication, and develop awareness of the impact of information systems on individuals and society; Problem solving topics of: problem formulation, statement of objectives, solution strategies, idea generation, solution implementation, and solution

validation were recommended to be examined and discussed in both individual and group situations; Design is described as being the activity that brings together all of the other areas together, most development activities involve either designing a program or require the developer to have an understanding of original design decisions (Freeman et al., 1976).

Freeman et al. (1976) provided an assessment of the state of software engineering education at the time the framework was proposed. They describe the beginning of the introduction of software engineering courses into the graduate level offerings of the computer science departments; commonly these courses tried to combine delivering theory principles (via lectures, discussions and readings), with application of the principles through a project (Freeman et al., 1976). These projects are often conducted in small groups and the emphasis is on the method used throughout the project rather than the end product (Freeman et al., 1976).

Ten years on Freeman (1987) wrote of the failure of most educational institutions, and the computing industry in general, to progress software engineering practices. Freeman (1987) describes some trends of the decade since he first proposed a framework for software engineering education:

- rapid expansion of the industry;
- creation of the software package industry as well as increasing custom development projects;
- microcomputer revolution leading to a demand for desktop program development;
- increasingly large software systems;
- failure by industry and educators to provide software engineering training; and
- very little attention paid to the internal technical design of programs

So far, not much had been achieved in terms of overcoming the software crisis described by Bauer (1973).

In an attempt to address the need for specialised software engineering education the Software Engineering Institute (SEI), part of Carnegie Mellon University, released an interim report on suggested content for a graduate level curriculum in software engineering in 1987 (Duggins & Thomas, 2002). This report defined 20 ‘content units’ that should be within a software engineering curriculum but made no attempt to sort them into courses; the content units were: the software engineering process, software evolution, software generation, software maintenance, technical communication, software configuration management, software quality issues, software quality assurance, software project organizational and management issues, software project economics, software operational issues, requirements analysis, specification, systems design, software design, software implementation, software

testing, system integration, embedded real-time systems, and human interfaces (Ardis & Ford, 1989; Duggins & Thomas, 2002).

A curriculum design workshop was then held in 1988 to take the suggested content units in the SEI report and design a model for a Masters of Software Engineering which was to be a "terminal professional degree", designed for the practitioner with a project or practicum element being used to demonstrate knowledge attained (Ardis & Ford, 1989, p. 8). The result of the workshop was a curriculum made up of six core courses, a project component, and electives making up 20 – 40% of the program (Ardis & Ford, 1989). The core courses were: software systems engineering, specification of software systems, principles and applications of software design, software project management, software verification and validation, and software generation and maintenance (Ardis & Ford, 1989). Electives were to come from the following recommended categories: software engineering topics, computer science topics, systems engineering topics, application domain topics, and engineering management topics (Ardis & Ford, 1989).

The experience component was suggested to be at least 30% of the student's work and could be offered in a variety of ways such as: a capstone project - where student completes a project after completion of a majority of coursework; a continuing project – where students are part of an in-house software factory that works on an on-going application development task; a multiple course coordinated project – where "a single project is carried through four courses (on software analysis, design, testing, and maintenance)"; an industry cooperative program – where students go out to industry for six months then return to complete their studies; a commercial software company – where students work in a commercial operation established by the university in cooperation with local industry; a design studio – where students work on a project in an apprenticeship type relationship under the guidance of an experienced developer (Ardis & Ford, 1989, p. 42).

The examples of how to provide experience to software engineering students were all currently in use at the 15 universities, who at the time, offered the only graduate software engineering programs (Ardis & Ford, 1989). The two most common approaches being the capstone project and the inclusion of a project in a lecture course (Ardis & Ford, 1989). One of the earliest reflections by researchers on the course integrated project experience comes from the University of Toronto's "Software Hut" first delivered in 1973 (Horning & Wortman, 1977, p. 325). Each software hut was a team of three tasked with design and creation of one of two software components; following this teams were required to purchase a module from other software huts and interface them with their own; a later phase involved modifying a system consisting of the two components built by other software huts and adding to the program specifications so changes to the software were required (Horning & Wortman, 1977). All of this was done under financial constraints with software huts earning 'program engineering dollars' based on the grade earned for each phases of the project, the money

then used to purchase components from other software huts for the following phase (Horning & Wortman, 1977).

The Association of Computing Machinery (ACM) and the Institute of Electrical and Electronic Engineers (IEEE) worked together in 1991 to produce a broad set of curriculum guidelines for any programs covering 'computing', a term they used to describe all of the different computer related fields (Duggins & Thomas, 2002). This curriculum covered nine content areas and included an area on software methodology and engineering; software methodology and engineering was to cover: problem solving concepts, software development process, software requirements and specifications, software design and implementation, and verification and validation (Duggins & Thomas, 2002).

Though there now had been good progress made to define what a graduate level course for software engineers should comprise, educational institutes were still sluggish to address the need for software engineering practices to be taught throughout undergraduate level programs. A 1994 report from the SEI states the researcher found no "undergraduate programs named bachelor of science in software engineering at any United States universities" (Ford, 1994, p. 3). Ford (1994) goes on to describe the efforts of eleven universities to address the need for an undergraduate software engineering curriculum. These institutes offered either a sequence of software engineering courses in their undergraduate computer science program or an undergraduate program that is software related (e.g. Bachelor of Science in Software Engineering Technology) (Ford, 1994). The former approach of a sequence of software engineering courses within an undergraduate computer science degree was the most prolific, with Ford (1994) stating that nearly all computer science programs now included at least one software engineering course.

So, it had become common practice to include one or more software engineering courses into existing undergraduate Computer Science programs to try to satisfy industry demands for skilled practitioners (Ford, 1994; Leventhal & Mynatt, 1987; Meziane & Vadera, 2004; Parnas, 1999). Ford (1994) speculates that the slow growth of the dedicated software engineering curriculums was due in part to the well-established computer science degrees providing programming graduates that were proficient enough for an inefficient industry. As the size and complexity of the systems being developed grew, so would the demand for skilled software engineers; the differences between the disciplines would become more apparent and the need for separate educational programs more urgent (Ford, 1994).

Borenstein (1992) describes the content of a typical computer science program as comprising: programming fundamentals; data structures – ways of storing and describing data; abstract data types – methods for describing a class of data structures and the operations that can be performed on the data; analysis of algorithms; program verification – mathematical proof of correctness of the algorithm; program synthesis –automatic generation of a program from its mathematical description; and computational mathematical

theory. He states that most of this content is of little use to the professional programmer, with the exception of programming fundamentals and data structures (Borenstein, 1992). The in-depth mathematical theory is overkill as programmers can implement the maths required for a specific task within a program with relatively superficial knowledge; in the same vein programmers are never required to verify mathematically the capabilities of their systems, this is done through practical testing methods (Borenstein, 1992).

To find out what a typical undergraduate software engineering course was comprised of, Leventhal and Mynatt (1987) surveyed a sample of USA and Canadian institutes offering a Bachelor's degree in Computer Science. They state that "the typical course focuses on the software development life cycle and involves a significant project worked on by teams of students, with student leaders" (Leventhal & Mynatt, 1987, p. 1197).

In regard to topics covered, three types of courses were identified: early life-cycle courses, later life-cycle courses, and theoretical issues courses (Leventhal & Mynatt, 1987). Early life-cycle courses cover topics in the early stages of the software development life cycle: requirements analysis – where user requirements are elicited; requirements specification – where the elicited requirements are defined in analysis models/diagrams; and system design – where the analysis models are changed to reflect technology decisions that are made before development begins (Leventhal & Mynatt, 1987). A large part of the student grade is based on a project and these courses make heavy use of written reports to document project outcomes, reflecting the actual deliverables of these phases in industry (Leventhal & Mynatt, 1987).

Later life-cycle courses were the most predominant of the course types and cover the phases of the software development life-cycle that produce functioning code: detailed design, coding, testing, and maintenance – where errors are fixed or additional functionality is added to the software (Leventhal & Mynatt, 1987). Leventhal and Mynatt (1987) state this type of course is product oriented with a large portion of the students grade related to a project. The later life cycle phases, where code is produced, have the most in-depth coverage of all of the topic areas in the software engineering courses and are perceived by faculty as being more effective than those focusing on the non-code producing elements of software engineering practice (Leventhal & Mynatt, 1987).

Leventhal and Mynatt (1987) describe the Theoretical-Issues course as the least commonly occurring course type, covering the topics of: software metrics (estimation), project management, ethical and legal concerns. This course is heavily theory based and aimed at higher level students; students rely on journal articles as the main course resource and are involved in self-grading of the project component (Leventhal & Mynatt, 1987).

Over 95% of the institutes surveyed included a project component in at least one of their software engineering courses; a feature that was common across all three course types (Leventhal & Mynatt, 1987). Projects were combinations of toy projects and ones intended

for actual use, with the lecturer role playing the user; a majority of institutes made use of teams at least some of the time, with 90% reporting that teams were used frequently (Leventhal & Mynatt, 1987). Most courses have the student teams work on different projects with only a small percentage engaging the whole class in one project; a majority of the project teams are student led and the project accounts for 40% or more of the student grade, with the lecturer normally having the entire responsibility for grading of the project (Leventhal & Mynatt, 1987).

Shaw and Tomayko (1991) describe five different models of project work used in one semester, undergraduate software engineering courses. All of the models assume students have knowledge on program construction from prerequisite courses, with the aim of the software engineering course being "to demonstrate how this technical material is applied in the context of large scale software development" (Shaw & Tomayko, 1991, p. 38). The five models (refer Figure 2.2) are: software engineering as artifact, topical approach, small group project, large project team, and project only (Shaw & Tomayko, 1991, p. 40).

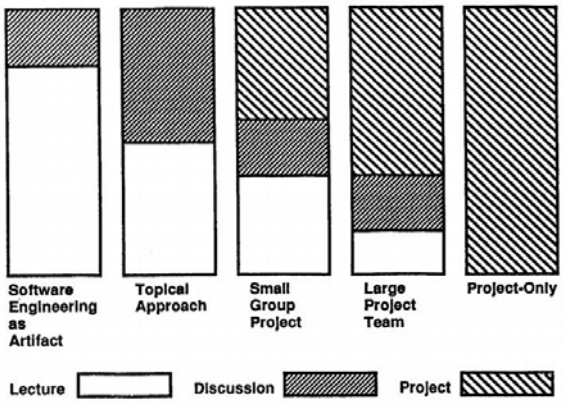


Figure 2.2: Models of the One-Semester course

Described as the "all-talk, no-action" models, the software engineering as artifact model and the topical approach model rely heavily on lectures and some discussion to teach software engineering concepts (Shaw & Tomayko, 1991, p. 39). These two approaches have the advantages of: being able to fit nicely into an academic semester or quarter, and allowing the students to focus on issues presented by the instructor rather than constantly troubleshooting project problems (Shaw & Tomayko, 1991). However, Shaw and Tomayko (1991) point out that two key areas that lead to project failure, communication issues and configuration control, cannot be appreciated until students experience working with others on a software product.

The small group project, the most common model in use at the time, splits the course evenly into project work and class work (Shaw & Tomayko, 1991). The project is limited in scope and size due to the constraints of it needing to be completed within a term by a group of between three to five students (Shaw & Tomayko, 1991). Shaw and Tomayko (1991)

describe this small team, small project, fake customer approach as really just an extension of "the programming-in-the-small" experience students are likely to have had during their computer science course work but they indicate that by paying particular attention to configuration management and quality assurance larger project issues can be taught (p. 39).

Shaw and Tomayko (1991) assert that if the course "is to immerse the student in a practical, real-life, software product development process" then the large project team model is the way to do it (p. 40). This approach has the students all take on different roles (e.g., designers, quality auditors, configuration managers) within one large project team to work on one software product; there is often a real customer and students learn about the other roles within the team through the normal interactions that happen during the project (Shaw & Tomayko, 1991). The project only model is similar but rather than spend any course time discussing software engineering topics, students learn entirely by being immersed in the project; this is a common approach of the capstone course (Shaw & Tomayko, 1991).

In 1993, the Association for Computing Machinery (ACM) and the IEEE Computer Society created a joint Steering Committee for the Establishment of Software Engineering as a Profession, five years later this became the Software Engineering Coordinating Committee (SWECC) (Duggins & Thomas, 2002). One of the main projects of this committee was to document an agreed upon core body of knowledge for the Software Engineering discipline, this became known as SWEBOK – the Software Engineering Body of Knowledge (Abran, Moore, Bourque, & Dupuis, 2004; Duggins & Thomas, 2002; Ludi & Collofello, 2001; Sinderson & Spirkovska, 2001). The elements of SWEBOK will be discussed further in section 2.4.1.

Another major project for SWECC was the Software Engineering Education Project (SWEEP); contributing to this project was the Working Group on Software Engineering Education and Training (WGSEET) who were the first to propose a model curriculum for a BS of Software Engineering, also applicable to a BS of Computer Science or Information Systems (Sinderson & Spirkovska, 2001). Guidelines for software engineering curriculums were formally defined by the joint ACM/IEEE committee in a report known as SE2004 (Lethbridge, LeBlanc, Sobel, Hilburn, & Diaz-Herrera, 2006; Schneider, Johnston, & Joyce, 2005). SE2004 identifies ten knowledge areas that are fundamental to the education of software engineers, these are commonly referred to as SEEK (software engineering education knowledge) (Lethbridge et al., 2006; Thompson & Reed, 2005).

The ten areas of SEEK are: computing essentials e.g., computer science topics of programming, databases, algorithms, operating systems, and human factors; mathematical and engineering fundamentals e.g., maths, statistics, trade-off analysis, and prioritisation; professional practice e.g., group dynamics, communication skills, ethics, and professional conduct; software modelling and analysis e.g., requirements engineering and modelling principles; software design e.g., patterns, architecture, human-computer interaction, detailed

design, tools, and evaluation; software verification and validation e.g., metrics, measurement, reviews, inspections, and testing; software evolution e.g., planning for evolution, reverse engineering, impact analysis, migration, and refactoring; software process e.g., lifecycle models, standards, individual processes, and process modelling; software quality e.g., quality attributes, costs and impacts of bad quality, and quality standards; software management e.g., planning, control, personnel and organization, and configuration and release management (Lethbridge et al., 2006; Thompson & Reed, 2005).

SEEK is outcome focused and identifies specific tasks software engineering graduates should be able to perform and behaviours they should possess. These included: being able to work in teams as well as individually; being able to make good decisions around system compromises necessary due to the common project constraints of time, cost, and existing systems; being ready for work due to mastery of software engineering knowledge and skills; being able to make ethical and economically sound design decisions in one or more domains; being able to apply current techniques and models appropriate for the phase of the systems development life cycle; having a commitment to lifelong learning necessary for the rapidly changing environment; being able to communicate; and being able to negotiate effectively as well as demonstrating effective work habits (Lethbridge et al., 2006).

Ford (1994) speculated that with the introduction of a software engineering model curriculum by professional societies would greatly increase the occurrence of these programs in universities. This appears to be the case as a few years later Sinderson and Spirkovska (2001) report a marked increase in the number of software engineering offerings worldwide.

With the formalising of the knowledge base for software engineering education, Meziane and Vadera (2004) decided to investigate the offerings of 44 English universities to determine the differences between their computer science and software engineering bachelor degrees by mapping the course content to SEEK topics. Bachelor degrees in the UK, similar to New Zealand and Australia, are of three or four years duration, with the four year degree programs embedding an industrial placement year, typically in year three (Meziane & Vadera, 2004). Meziane and Vadera (2004) state “there is very little difference between the SE and CS programs currently offered in English Universities” (p. 69).

The differences between the two types of programs were in the following areas: software modelling and analysis were emphasised more in the software engineering programs; intelligent systems, present in nearly all of the computer science programs, were not taught in most of the software engineering programs; neither program types contained much in the way of mathematics, reflecting the decreasing number of students taking maths in the final years of high school (Meziane & Vadera, 2004). Meziane and Vadera (2004) do not believe this is due to there being no clear model for software engineering education until recently, rather it is mainly a result of economics. Computer science programs are well-established and software engineering program enrolments are small in number, meaning it is more

viable to have small variations on the content of computer science programs and deliver it as a software engineering program (Meziane & Vadera, 2004).

Swinburne University of Technology was the first university in Australia to offer a Bachelor of Software Engineering; introduced in 1997 the degree has been reviewed and revised regularly and is based substantially on the suggested curriculum within SE2004, the report outlining SEEK (Schneider et al., 2005). Schneider et al. (2005) surveyed graduates of the Bachelor of Software Engineering to determine how relevant the areas covered during their education were to their current positions and their perception of how well the degree prepared them to work. The majority of the respondents reported feeling confident entering employment and that their employers quickly recognised their capability and gave them more responsibility (Schneider et al., 2005). Areas where graduates felt their skills has been developed well during their studies included: coding, requirements elicitation, teamwork, quality assurance, and design; areas requiring improved coverage were: project management, risk management, and conflict resolution (Schneider et al., 2005). Schneider et al. (2005) discuss that Engineers Australia has an accreditation requirement that a Software Engineering degree contain an engineering component, exposing students to the body of knowledge of another engineering discipline; graduates rated the minor engineering stream as not being useful, or valued by their employers.

When asked to consider the importance of the SEEK knowledge areas, the graduates rated mathematics and engineering fundamentals as being the least important area; SEEK puts a large emphasis on this area ranking it as the second most important (Schneider et al., 2005). The graduates saw software quality as being more important than SEEK ranks the area; but computing essentials, the most emphasised SEEK area, was agreed to be the most important by the graduates (Schneider et al., 2005).

2.4 Skill requirements of junior developers – An industry perspective

Industry has found that graduates are not lacking in technical skills, they are lacking in people and process skills (Hilburn, 1997). As new employees they: struggle to communicate effectively, have little experience in team work, are unable to effectively manage their workload, lack appreciation of organisational structures, and have little understanding of business processes (Hilburn, 1997). A recent investigation conducted by the Australian Learning and Teaching Council reports that employers find it can take anywhere from three to 12 months for “graduates to get up to speed in industry”, often costing companies more than $10,000 per graduate (Koppi & Naghdy, 2009, p. 8).

In 1983, a company called Plessey Telecommunications (later renamed GPT and now known as Siemens GEC Communication Systems Ltd) started an in-house training course for their computer science graduates with the aim of highlighting “the differences of working

in the real world compared with the near 'ideal' environment experienced at university" (Dawson et al., 1997, p. 287). The course was instigated by the companies software managers who were finding that new graduates took up to six months to become productive; the productivity delays were not only to do with the graduates' learning curve around the company's tools and product lines, it also involved their lack of readiness to deal with the realities of the workplace (Dawson, 2000). New employees participate in a two week group project as part of a team of four or five; the first day involves a set up session where the project aims are outlined and graduates are made aware of difficulties that are often faced in real software engineering projects; the final day includes a review of how the project went with the remaining time being devoted to the development task (Dawson, 2000).

During the two week project the course instructor role plays the customer, manager, and other key personnel, such as a quality auditor, and a number of what Dawson (2000) refers to as "dirty tricks" are played on the teams (p. 210). Dawson (2000) describes these dirty tricks as being the defining difference between the real-life project experiences that graduates may have had during their education and the real-life experience they are given at the in-house training course. He describes the educational approach of trying to provide the best possible environment for students to learn in as being detrimental, with students having an unrealistic view of what is involved in the real world of software development (Dawson, 2000).

Dawson (2000) outlines the 20 dirty tricks that have been used in various combinations during the in-house training course to simulate real work conditions: give them a vague initial specification with ambiguous statements and missing detail; make all their assumptions incorrect, in particular reject any additional features they have added; role play a customer who does not know what they want and has low computer literacy; change the requirements, or re-prioritise them, regularly; have conflicting requirements; provide customers with conflicting ideas; have customers with different personalities (e.g., super enthusiastic vs. super reluctant); ban overtime; add tasks so the schedule is disrupted; move deadlines; have quality auditors schedule inspections randomly; change the truth slightly and then deny that there has been any change; swap team members around; change the work practices to simulate changes in management; upgrade software mid-project; change the hardware platform that the software will run on towards the end of the project; crash hardware; slow down the development software by overloading the network so builds and testing take twice as long; delete files so teams have to revert to the last backup; finally, say I told you so when the graduate developers express their frustration.

Of these 20 dirty tricks about half are used during a course with only the inadequate specification and regularly changing requirements always being included (Dawson, 2000). Over the time this training course has been run the company has seen a steady improvement in the work readiness of graduates with more institutes providing real-life

project experiences; however, they believe there is still significant work to be done before the in-house training course becomes redundant (Dawson et al., 1997).

Brechner (2003), in charge of developer training at Microsoft Corporation, states that new developers have to go through months of in-house training before they are "trusted to write new code" (p. 134). He proposed five new courses that Computer Science students should take to narrow the gap between what is learnt in school and what graduates need to know to work in industry (Brechner, 2003). The first course was design analysis, where students not only learn how to apply a design pattern (a reusable solution to a common problem) but also are able to look at existing code and improve it by making it more readable and maintainable; secondly; it was deemed necessary to introduce a course on globalisation and accessibility so that graduates are better prepared to develop software for a diverse range of users; to address the difference between academic based projects and commercial ones a course with a multidisciplinary (e.g., coders, designers, technical writers) project team that has real users, vague requirements, and deadlines is seen as a necessary course; a large scale development course would address the need for graduates to be able to write code that will be integrated into a larger piece of software, developers need to be able to read, modify, and debug the code of others who are working on the same project; finally, a course on writing quality code that has a long life is required to teach students about filtering user input, and controlling resources through access control and cryptography (Brechner, 2003).

In 2008, an observational study undertaken by Begel and Simon (2008b) on new graduates in software developer roles at Microsoft discussed that while the graduates were proficient in: coding, reading and writing of design specifications, and problem solving; they experienced difficulties in the areas of: communication, collaboration, technical tools used for large-scale development, cognition, and orientation within their project teams where there may be little in the way of well organised information. The tasks undertaken by novice software developers during the study were categorised as presented in Table 2.2 (Begel & Simon, 2008a, p. 5).

Table 2.2: Tasks of novice software developers

Task	Subtask
Programming	Reading, Writing, Commenting, Proof-reading, Code Review
Working on bugs	Reproduction, Reporting, Triage, Debugging
Testing	Writing, Running
Project Management	Check in, Check out, Revert
Documentation	Reading, Writing, Search
Specifications	Reading, Writing
Tools	Discovering, Finding, Installing, Using, Building
Communication	Asking questions, Persuasion, Coordination, Email, Meetings, Meeting prep, Finding People, Interacting with Managers, Teaching, Learning, Mentoring

Begel and Simon (2008a) found communication was the highest frequency activity for the novice software developers, and was deemed to be critical to productivity. The novices

communicated to: coordinate activities, ask for help, participate in meetings, report to managers, work with others, get feedback, give help, find people, persuade others (Begel & Simon, 2008a). Novice software developers encounter difficulty over knowing when and of whom to ask questions, often showing reluctance to ask for help early so as not to appear incompetent (Begel & Simon, 2008b). Collaboration within large teams and working alongside multiple teams were major areas of difficulty and uncertainty for the novices; some found themselves being given tasks by co-workers which were not actually set by the manager but still undertook them without complaint (Begel & Simon, 2008b).

Documentation was the next most frequent activity, often combined with programming and debugging tasks (Begel & Simon, 2008a). Documentation tasks involved: searching of documentation to aid in understanding whilst debugging or creating code; writing of bug reports and specification plans; writing personal notes for meetings; recording of information around tool use for later reference (Begel & Simon, 2008a). Novices expressed concern over how to structure and record the personal notes in a way that would be accessible later (Begel & Simon, 2008a).

Working on bugs and programming were the next highest frequency tasks, respectively; both tasks often in conjunction with documentation being used (e.g., to help understand what's happening in the codebase, or to aid with understanding an API, or when replicating an online sample of code) (Begel & Simon, 2008a). Novices found difficulties in searching and navigating their way through the codebase during debugging, relying heavily on comments to try and understand the code; this led to them being very conscientious code commenters often adding comments on lines of code they were not actually altering for the fix (Begel & Simon, 2008a).

The novice software developers were regularly using tools or engaged in project management tasks, the activities included: revision control systems, compiling the project, setting up the development environment or their project within it, running auxiliary tools, tool creation to support project management tasks (Begel & Simon, 2008a). Depending on the phase of the development lifecycle that the novice software developer's team was in, the level of interaction with the specification and the amount of testing conducted varied greatly (Begel & Simon, 2008a).

Begel and Simon (2008b) summarise the major difficulties experienced by the novice software developers under the following categories: communication, collaboration, technical, cognition, and orientation. The novice developers suffered communication issues around: knowing when and how to ask questions; asking questions with the appropriate amount of information for the circumstance (Begel & Simon, 2008b). Novice developers who were not native English speakers struggled with abbreviations being pronounced as words, and with words with different meanings but the same pronunciation, often leading to miscommunication (Begel & Simon, 2008b). Collaboration problems were around: adapting

to working with a large team, interaction with other teams working on the same project, and working with a large existing codebase (Begel & Simon, 2008b).

Begel and Simon (2008b) observed technical issues such as: correct use of the revision control system, and lack of robust testing due to not having access to the environment in which tests were executed. The use of Visual Studio's (Microsoft's development environment) debugger to execute breakpoints, a key debugging technique, was seen as critical during the debugging tasks; some novices demonstrated advanced technique and could navigate through the large codebase without referring to documentation, they reported having been "clued in to certain debugging techniques" (Begel & Simon, 2008b, p. 228). The other novices struggled with documentation to find the part of the code to debug; frequently taking a long time to find information, and when information was discovered they would often find it was out dated (Begel & Simon, 2008b). The technical difficulties were often seen alongside collaboration and orientation problems (Begel & Simon, 2008b).

Novice software developers have large challenges around cognition; an enormous amount of information was presented to new employees, collecting and organising this information so it was useful later on was a difficult task; some novices emailed themselves notes, others extracted important task related information from emails into other documents; sometimes impromptu teaching sessions would occur when a co-worker was approached for information, novices found taking notes difficult in these situations; there is a reluctance to press for more information if the novices are not following an explanation or lack full understanding when given an answer to a question, this was interpreted as them being anxious to not waste a senior worker's time (Begel & Simon, 2008b).

Orientation problems with team membership, codebase, and organisational resources were exacerbated for the novice software developers by often badly organised documentation, leading to much frustration and lack of progress; novices often did not know everyone on their team and were unsure who to talk to over particular issues; even though a mentoring program existed one novice had to request one after four weeks and ended up with a mentor who was too busy to provide quality information (Begel & Simon, 2008b).

Begel and Simon (2008b) describe some common misconceptions that hindered the novice software developers during the study:

1. to make the manager happy, a novice developer should do everything themselves;
2. a novice developer who finds a bug should fix it, and should fix it perfectly, even if time is limited;
3. everything would be fine if there was more documentation; and
4. a novice developer knows when they are stuck.

Begel and Simon (2008a) discuss that while educational institutes endeavour to prepare graduates for industry by: teaching core computing theories that will help their learners keep apace with industry developments, and updating course content to reflect the current technical skills required by employers; there has not been enough focus on the "soft" skills required to tackle the human elements of software engineering (p. 3). They descirbe the human elements of software engineering as being:

> "the ability to create and debug specifications, to document code and its rationale and history, to follow a software methodology, to manage a large project, and to work with others on a software team" (Begel & Simon, 2008a, p. 3).

In a typical software engineering course, a team of three to five students are given a set of stable requirements, from a 'real' customer, then design and build a software product to meet those requirements by the end of the course; the point being to simulate a "greenfield", new software development, project so students are exposed to a variety of development phases and learn to interact in a team (Begel & Simon, 2008a, p. 12). Begel and Simon (2008b) point out that this type of project does not reflect the situation that most novice developers will find themselves in, joining a large pre-existing team, as the most junior member, and spending most of their time, initially, working on debugging a large pre-existing codebase.

In 1997, Lethbridge (1998a) surveyed 168 software developers to find out how relevant their education had been. The survey asked participants to rate 57 topics to discover: the extent to which the topic had been covered in their education, how much the participants now know (to identify forgotten knowledge or knowledge gained on the job), the importance of the topic for their role, and those topics the participants wished to learn more about (Lethbridge, 1998a). The topics were presented under the following general categories: 31 software topics, 9 mathematical topics, 4 engineering (hardware) topics, and 13 other topics taken from natural and social science, and business electives (Lethbridge, 1998b). A majority of the respondents to the survey were from Canada (74%) and the USA (23%); 50% had computer science or software engineering qualifications, 30% studied computer or electrical engineering with the remainder scattered over a variety of other fields; and 28% were classed as juniors having less than four years on the job (Lethbridge, 1998a).

Of the 31 software topics, respondents felt as though they had not learned the basics during their education for over half of them; some of this can be attributed to not taking electives (e.g. pattern recognition, graphics) and the newness of some of the topics (e.g., object oriented analysis and design). But some core topics, such as user interfaces and project management, appeared to have not been covered sufficiently during their formal education (Lethbridge, 1998a). The greatest on the job learning occurred in the following areas: configuration management, testing and quality assurance, process standards, maintenance and reengineering, project management, object oriented analysis and design, user

interfaces, and requirements gathering (Lethbridge, 1998a). The topics which the respondents thought had too much emphasis during their education and had been the least important to them in their careers were the more theoretical or mathematical topics (e.g., calculus, numerical methods, artificial intelligence, pattern recognition) (Lethbridge, 1998a).

Lethbridge (2000) sought to improve on the survey conducted in 1997 by: redesigning the instrument to cover 75 topics obtained from common university curricula and topics appearing in the proposed SWEBOK; broadening the participant sample so there was less bias (software professionals came from a variety of industries and represented 24 countries); and refining the questions so that instead of the fourth question asking what topics they wished to learn more of, they now were asked to rank "how much influence has learning the material had on your thinking (your approach to problems and your general intellectual maturity), whether or not you have directly used the details of the material?" (p. 45).

The responses to the questions asking how influential and how useful the topic has been were deemed to show the importance of the topic for a software professional's career; the general software design topics were all ranked high in importance (data structures, algorithm design, software design and patterns, software architecture, object oriented concepts, and specific programming languages) (Lethbridge, 2000). Other software engineering topics that ranked highly included: requirements gathering and analysis; analysis and design methods, testing, verification and quality assurance; project management; configuration and release management; human-computer interaction/user interfaces; and databases (Lethbridge, 2000). The least important topics were in the continuous mathematics, electrical and computer engineering, and natural science categories; the non-technical topics that were ranked the most important were: ethics and professionalism, technical writing, leadership, and presenting to an audience (Lethbridge, 2000).

In terms of topic coverage during their education, the general software design topics were the only software engineering topics taught extensively, most other topics that received thorough coverage were in the computer science theory and mathematics categories; the topics where participants felt they had learned the least were software management, business, and people skills (Lethbridge, 2000). Lethbridge (2000) proposes that rather than courses that focus on continuous mathematics and basic sciences, software engineering curricula should put more emphasis on courses that teach people skills (leadership, negotiation), software processes, human-computer interaction, real-time design, and management.

Van Slyke, Kittner, and Cheney (1998) surveyed employers from a variety of industries to determine the skills they thought most necessary for their IT graduate workers to possess. Employers in this sample placed less importance on specific technical skills, overwhelmingly ranking more highly the general skills of: written communication, analytical thinking, general thinking, team work, listening ability, and self-motivation (Van Slyke et al., 1998). The IT

related skills that were considered most important for entry-level workers were the more general ones, such as: systems analysis and design, database concepts, general programming techniques, and systems testing (Van Slyke et al., 1998).

Several Australasian studies, involving samples of IT employers or IT employees, have identified similar themes in terms of important skills and skill gaps of IT graduates (Bekesi & Gardner, 2003; Koppi & Naghdy, 2009; Snell et al., 2002; Young et al., 1999). Young et al. (1999) asked employers to rank how important a selection of technical skills were, and to comment on areas where graduate skill sets were lacking. The top ranking technical skills were: systems analysis and design, specific programming languages, user training and support, operating systems, databases, networking, systems administration, use and evaluation of software packages, information gathering, and project management (Young et al., 1999). When asked to identify areas where urgent up-skilling was required the respondents overwhelmingly highlighted interpersonal and management skills, these included: written communication, ethical consultation, time management, team membership, presentation skills, information finding, customer service, and general people skills (Young et al., 1999).

Bekesi and Gardner (2003) asked of software developers how important certain technical skills were for their roles when hired, and how important those skills were currently. Of the 19 categories in the survey, the top ranking skills in order of importance when hired were: testing, specific programming languages, use of Microsoft Office and email, Graphical User Interface design, relational databases, and systems analysis and design; the current top ranking skills, in order of importance, were: systems analysis and design, specific programming languages, information gathering techniques, client consultation skills, project management skills, testing, use of Microsoft Office and email, Graphical User Interface design, and ergonomics (Bekesi & Gardner, 2003).

Snell et al. (2002) asked a sample of IT employers and recruitment agencies to rank the importance of soft skills for IT roles. Nineteen non-technical skills were rated; nine were defined as interactive skills (listening, interpersonal, relationship building, written communication, adaptability, teamwork, friendliness, attire, and grooming) with the remaining ten being defined as motivational skills (planning, initiative, problem solving, enthusiasm, stress tolerance, dependability, time management, innovation, willingness to learn, and self-confidence) (Snell et al., 2002). Though all skills were ranked as important, the three top interactive skills were listening, interpersonal, and teamwork; the four top motivational skills were willingness to learn, enthusiasm, initiative, and problem solving (Snell et al., 2002).

More emphasis is being placed on software maintenance skills as the software industry shifts from having lots of new product development work to having more reuse of existing systems, adapted to suit new requirements (Kajko-Mattsson, Forssander, Andersson, & Olsson, 2002). In summarising the literature, Kajko-Mattsson et al. (2002) describe software

maintenance as requiring more than half of the resources allocated to the life-cycle of a software product. Gunderman (cited in Kajko-Mattsson et al., 2002, p. 57) states that "maintenance has been viewed as a second class activity, with an admixture of on-the-job training for beginners and low-status assignments for the outcasts and the fallen". System maintenance tasks are often assigned to the inexperienced, who lack formal training in this area, consequently when they make what they deem to be small changes to code it can often lead to severe damage of the whole system (Kajko-Mattsson et al., 2002). McQuire & Randall (cited in Kajko-Mattsson et al., 2002, p. 58) assert "a highly skilled maintainer is the most important organisational asset pivotal for achieving quality software, strategic for improving maintenance and development processes, essential for remaining competitive and critical for business survival". They argue that a maintainer should be a skilled diagnostician and that it is up to educational institutes to prepare students for these roles (Kajko-Mattsson et al., 2002).

The Australian Learning and Teaching Council surveyed both employers and graduates employed in ICT positions to determine if Australian ICT curricula were developing work-ready graduates (Koppi & Naghdy, 2009). Both groups identified the following areas as being deficient in ICT graduates: communication skills, general awareness of the business environment and what is required in a job role, and problem solving abilities (Koppi & Naghdy, 2009). As well as these commonalities, employers highlighted the following interpersonal skills as needing improvement: initiative, self-management, independent learning, and planning (Koppi & Naghdy, 2009). Employers did not consider team-work as an area of deficiency, curricula appear to now be providing enough team-based experienced to enable graduates to quickly adapt to team-work in the business environment (Koppi & Naghdy, 2009).

Koppi and Naghdy (2009) recommend the following teaching practices to address the areas that graduates identified as needing improvement in ICT curricula:

- demonstrate subject relevance;
- have interactive sessions with students;
- use real-world examples and case studies;
- keep up-to-date with technology changes;
- provide group work related to industry practices; and
- design meaningful problem-solving activities (p. 10)

2.4.1 Software Engineers Body of Knowledge (SWEBOK)

As part of the effort to have software engineering recognised as an engineering discipline and a profession, SWECC (refer section 2.3.1, p 20) created the guide to the Software Engineers Body of Knowledge (SWEBOK) with the aim of providing "a topical guide to the

literature describing the generally accepted knowledge within the discipline" (Dupuis, Bourque, & Abran, 2003, p. 19). The Project Management Institute (cited in Abran et al., 2004, p. 24) describes generally accepted knowledge as knowledge that "applies to most projects most of the time, and widespread consensus validates its value and effectiveness". The guide was developed following five objectives:

1. to promote a consistent view of software engineering worldwide;
2. to clarify the place – and set the boundary- of software engineering with respect to other disciplines such as: computer science, project management, computer engineering, and mathematics;
3. to characterize the contents of the software engineering discipline;
4. to provide a topical access to the Software Engineering Body of Knowledge; and
5. to provide a foundation for curriculum development and for individual certification and licensing material (Abran et al., 2004, p. 22).

To ensure worldwide acceptance, as per the first objective, the guide went through a multi-phased implementation with the initial development phase involving 500 reviewers from 42 countries (Abran et al., 2004). The trial version that resulted was put out in 2001 for further consultation involving 120 reviewers from 21 countries, leading to a published edition in 2004 (Abran et al., 2004; Dupuis et al., 2003). The editors of the guide avoided focusing on fast changing technologies (e.g., specific programming languages, databases, networks) noting that, as with other engineering disciplines, the workforce outlives the technology; "an engineer must be equipped with the essential knowledge that supports the selection of the appropriate technology at the appropriate time in the appropriate circumstance" (Abran et al., 2004, p. xxvii).

The guide divides SWEBOK into ten knowledge areas: software requirements, software design, software construction, software testing, software maintenance, software configuration management, software engineering management, software engineering process, software engineering tools and methods, and software quality (Abran et al., 2004). These knowledge areas list topics and describe the depth of knowledge (using Bloom's taxonomy) that a software developer should have in these topics after four years experience (Ludi & Collofello, 2001; Schneider et al., 2005). The guide is seen as an evolutionary product and has a variety of intended audiences: it provides industry with ways of classifying jobs, specifying development tasks, creating performance evaluation criteria, and identifying training requirements; it provides academia with the areas and levels of knowledge requiring coverage in curricula and to award degrees; it provides professional societies and licensing bodies with guidelines for individual certification (Dupuis et al., 2003; Schneider et al., 2005). In *SWEBOK guide: An overview of trial usages in the field of education* Dupuis et al. (2003) describe a variety of applications of SWEBOK since the trial version release. SWEBOK knowledge areas are being referenced in many courses worldwide; multiple conference

papers and calls for papers include workshops, evaluations or critiques of SWEBOK; SWEBOK has been used for curriculum design and evaluation both in academia and industry (Dupuis et al., 2003).

Frailey and Mason (2002) discuss how due to the nature of technology, and the newness of software engineering as a profession, best practices are replaced several times during a developer's career. They note that in many software development organisations there is a lack of professional development in all areas except developer tools training, due to this there is a likelihood that many software organisations will fall behind as older practices become entrenched (Frailey & Mason, 2002). The Securities Industry Automation Corporation (SIAC) is a software development organisation who build and operate floor trading systems used in the New York Stock Exchange, they invest heavily in professional development for their employees offering both leadership and technical training (Frailey & Mason, 2002). When SIAC used SWEBOK as a tool to help them identify software engineering roles and tasks likely to be undertaken by practitioners in those roles, the workforce reacted positively and could easily identify themselves as working within one or more knowledge areas (Frailey & Mason, 2002). SIAC then proceeded to use SWEBOK to identify competencies within those knowledge areas, leading to design of education programs specific to particular roles; employees found it easy to select programs that matched their needs in their current role, or programs that developed their skills in other areas they wished to move into; managers were more easily able to assess their teams' capability and identify gaps in skill sets (Frailey & Mason, 2002).

Southern Methodist University (SMU) utilised SWEBOK to help them improve their masters level software engineering program, acknowledging that keeping programs relevant was challenging in an opportunistic funding environment where faculty often have specialised interests (Frailey & Mason, 2002). SMUs' masters, first introduced in 1993, has undergone regularly updating and revising using industry input and models such as: the Software Engineering Institutes Capability Maturity Model, ISO 12207, and more recently SWEBOK (Frailey & Mason, 2002). The common terminology of SWEBOK allowed SIAC and SMU to easily identify areas where SIAC professional development requirements could be met by courses provided at SMU, allowing SMU to create a series of six certificate programs (a core software engineering certificate and five speciality certificates) that SIAC employees can complete, gradually building up their knowledge and allowing them to go on to finish the masters in a manageable way for full-time employees (Frailey & Mason, 2002).

Arizona State University used SWEBOK to assess the relevancy of their introductory software engineering course (Ludi & Collofello, 2001). This course follows the standard approach of a team of students working together on a greenfield project (i.e., a new development project with no need to consider previous work) from requirements definition to acceptance testing of the finished software product (Ludi & Collofello, 2001). The researchers used Bloom's taxonomy to map the course topics and objectives to the

appropriate SWEBOK knowledge areas, which highlighted the areas where students were not receiving sufficient knowledge or experience.

Whilst it is useful to evaluate a course's content against industry skill requirements Lethbridge et al. (2006) point out that there is a significant difference between the skills described in SWEBOK, focused on a practitioner with four years experience, and SEEK the software engineering education outcomes that describe what a graduate should know at the end of their studies. SWEBOK is of greater depth, SEEK has broader coverage incorporating topic areas such as: computer science, mathematics, project management, and human computer interaction; SWEBOK omits these areas, seeing them as related disciplines rather than core software engineering concepts (Lethbridge et al., 2006).

2.5 Modern adaptions to traditional Software Developer education

In order to try and overcome the deficiencies within existing curricula, educators have adopted and implemented many evolving best practices within coursework. In particular it is common to hear about the use of: the Software Engineering Institute's Personal Software Process (PSP), agile process techniques in development (in particular the use of pair programming techniques to improve code quality) and requirements analysis, role-play and/or simulations, enterprise level tool use (e.g., large database systems, team development tools), and team development processes (Kessler & Dykman, 2007; Maletic, Howald, & Marcus, 2001; Postema, Dick, Miller, & Cuce, 2000; Reichlmayr, 2003; Sherrell & Robertson, 2006; Simon & Hanks, 2008; Suri & Sebern, 2004).

2.5.1 Personal Software Process

The PSP is a framework, devised by Watts S. Humphrey of the Software Engineering Institute at Carnegie Mellon University, which provides software engineers with tools that enable them to measure their productivity, and improve on the processes they use to work individually and in teams (Humphrey, 1996). The PSP introduces the following software engineering methods: data gathering, estimating of software size and resources required, management of defects (bugs), productivity analysis, and yield management (Humphrey, 1996). The primary measure of productivity within the PSP is lines of code (LOC); developers capture data around how many lines of code have been produced, and how many defects have been introduced into the product during a timed development session (Humphrey, 1996). The PSP moves through some evolutionary levels: PSP0 is described as personal measurement where the practitioner is undertaking time and defect recording and is working to a coding standard; PSP1 involves personal planning with the practitioner being able to estimate and undertaking task and schedule planning; PSP2 is about personal quality with the practitioner undertaking code and design reviews as well as tracking defects within the product; PSP3 is known as cyclic process where the practitioner is able to scale up to large system development (Humphrey, 1996).

Maletic, Howald, and Marcus (2001) incorporated part of the PSP into a graduate-level software engineering course at the University of Memphis as a means to introduce the ideas of: personal process improvement, estimating work effort, and the usefulness of software metrics (measurement). They discuss the difficulty of covering the entire PSP within a software engineering course in a computer science degree; as the amount of software engineering topics covered is already significant; the PSP is delivered within the SEI as a series of 15 lectures of around 90-120 minutes duration, these lectures are supported by 10 small programming tasks that are worked on at the rate of one per week (Maletic et al., 2001). They utilise five weeks of the 14 week semester to cover the major aspects of the PSP, providing enough detail so students are able to use some of the PSP forms for recording development effort and defects on their design and programming assignments, rather than recording data based on the 10 programming tasks (Maletic et al., 2001). Students were asked to evaluate the PSP at the end of the course; a majority of the students found learning the PSP easy but applying it was time consuming due to all of the manual forms that needed to be filled in; all students agreed that the forced record keeping helped them improve estimations of time, effort, and size as well as the overall quality of the software product (Maletic et al., 2001).

Suri and Sebern (2004) teach the PSP to undergraduates at the Milwaukee School of Engineering in order to introduce the students to practices which lead to quality software early in their studies (the course occurs during their second year of study). The initial delivery of the course involved students being moved from PSP0 through to PSP2 in increments, this was found to be highly distracting for the students with comments made as to how the focus of the paper became to learn about the changing process, rather than on tracking and measuring their data (Suri & Sebern, 2004). To address student concerns, further deliveries of the course used a modified process that was taught throughout (similar to the content of PSP2) rather than the processes changing during the course and spreadsheets were developed for data recording that automated the calculation of the derived measures (e.g., defect density, yield) (Suri & Sebern, 2004).

Suri and Sebern (2004) describe the following challenges in teaching the PSP: students find the recording of data to be burdensome and boring; getting students used to using the spreadsheets involved continual reiteration of the measurement concepts of “base and reused lines of code”; not enough programs are written to allow the students to fully attune their personal processes; students need constant reminding to reflect on the data that is being collected (p. 20). They feel these challenges are outweighed by the benefits obtained: students come to the point where they can see their productivity does not suffer from more complex programs and increased recording of data, in fact quality of their programs also improves; students were able to see that becoming a better developer was a matter of applying simple and practical techniques; becoming better estimators allowed the students to

delay starting assignments and still meet deadlines, this had the effect of them feeling that the skill of accurately estimating effort was very worthwhile (Suri & Sebern, 2004).

Postema, Dick, Miller, and Cuce (2000) integrated the PSP into a core second year paper, Software Engineering Practice, of Monash University's Bachelor of Computing degree. To overcome the laborious recording requirements of the PSP, a tool was created that would automatically produce the time and defect reports, as well as the project summaries (Postema et al., 2000). Use of the tool has enabled the students to focus more on the process, rather than the recording of metrics, and has assisted the lecturing staff in monitoring team progress, helping ensure a fair level of contribution from all students on team development tasks (Postema et al., 2000).

PSP has decreased in popularity as development techniques have evolved. The measurement of LOC, even though it is a standard, widely used software size metric, has multiple issues as a measure of software productivity: the number of lines of code required to implement functionality would differ between development languages; counting lines of code does not measure value-added functionality (advancement towards completing the things the customer wants in the product) or the knowledge required to implement the functions (Armour, 2004; Maurer & Martel, 2002). LOC within an application would decrease as the modern practice of refactoring was implemented (Maurer & Martel, 2002). Martin (cited in Moser, Abrahamsson, Pedrycz, Sillitti, & Succi, 2008) defines refactoring as "a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behaviour" (p. 252). Refactoring is standard practice within agile development methods and often results in an increase in the number of methods within the application, but a decrease overall of the total number of lines of code (Moser et al., 2008).

2.5.2 Agile development methods

The researcher became interested in Agile development methods in 2002, when tasked with introducing a group assessment into an undergraduate programming course (Cleland, 2003). Agile (also known as Adaptive or lightweight) development methods were receiving a large amount of attention at this time. This was partly due to the reported high rate of failure and delay occurring with traditional waterfall lifecycle development projects, where user requirements are defined early in the project and it can be a long time before the end user sees the software product (The Standish Group International, 1995). In a 2002 article, Jim Highsmith comments "While interest in agile methodologies has blossomed in the past two years, their roots go back at least a decade. Teams using early versions of Ken Schwaber's Scrum, Peter Coad's Feature-Driven Development and [my] Adaptive Software Development were delivering successful projects in the early to mid-1990s" (p. 30).

The dynamic nature of organisations, and the environments in which they operate, means a rigid long-term planning approach for IT development projects is unlikely to lead to a successful implementation. Goals and requirements identified at the beginning of the project may change dramatically over its lifetime. Friend and Hickling (1987) state "planning is viewed as a continuous process: a process of choosing strategically through time" (p. 1). The larger the organisation the more complex this becomes as collective choice comes in to play; different points of view need to be considered before common goals can be agreed upon (Friend & Hickling, 1987). Fauldi (1973) discusses two types of planning approaches: blueprint planning, where a detailed plan can be set at the beginning of a project due to the certainty of a thoroughly understood problem and stable environment; and process planning, where uncertainty of the environment and complexity of the problem means that planning must be adapted based on feedback during the project. The nature of technology means that certainty is not an aspect normally associated with an IT software development project. Users may be unaware of exactly how technology can assist their daily tasks and organisational environments are rapidly changing, both of these factors can mean the requirements for the project may change dramatically overtime. Disjointed incrementalism has been advocated as a process planning approach where long term goals are set but the steps taken to achieve these goals are entirely flexible (Fauldi, 1973; Reeve & Petch, 1999).

This concept is one that has been embraced by the IT software development community. What began as incremental and iterative development in the 1960s at NASA, became known as Rapid Application Development (RAD) in the 1980s, and Agile Development from the late 1990s (Larman & Basili, 2003). The incremental and iterative development (IID) approaches work on the principal of obtaining constant feedback from users to determine future requirements and assess usability. IID approaches are excellent at coping with uncertainty. No matter which approach is chosen (e.g., extreme programming (XP), SCRUM, RAD) they all begin with a high level overview of requirements, the establishment of the long term project goals, and then move into short delivery cycles (Larman & Basili, 2003). This means the project has the flexibility to adapt to any changes that may arise along the way. Users are consulted regularly and prototypes are used to elicit further requirements and gain user acceptance of functionality. Features are planned and developed in short iterations so that the users are seeing functionality early in the project and therefore stay enthusiastic and committed.

In February 2001, a group of likeminded individuals who were practicing IID techniques got together to agree upon the core principles of these techniques, the group dubbed themselves the Agile Alliance and the core principles became known as the Agile Manifesto (Highsmith & Fowler, 2001). The Agile Manifesto states:

> We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

That is, while there is value in the items on the right, we value the items on the left more (Highsmith & Fowler, 2001, p. 30).

The methodologies originally covered by the Agile Alliance were: Adaptive Software Development (ASD), Crystal, Dynamic Systems Development Method (DSDM), Extreme Programming (XP), Feature Driven Development (FDD), and Scrum (Hislop et al., 2002).

In 2002, the researcher introduced team-based development to students by teaching them a combination of two popular agile techniques: the SCRUM product development method and selected XP engineering practices (Cleland, 2003). Dubbed XP@SCRUM by agile practitioners Mar & Schwaber (2002), the combination of these two agile methods provided students with a strong framework to support their team-based development project as well as techniques which allowed for collaboration and frequent communication with users to obtain feedback on the software developed (Cleland, 2003). Teaching of Agile processes, and in particular the engineering practice of pair programming, has been adopted by many educational institutes looking to introduce students to modern development techniques. Pair programming requires two developers to sit side-by-side and work simultaneously on the code, one developer controls the keyboard and mouse while the other reviews the code looking for logic and syntax errors (McDowell, Werner, Bullock, & Fernald, 2006).

Kessler and Dykman (2007) describe teaching software engineering students at the University of Utah both traditional and agile processes. With the two approaches being widely practised in industry, Kessler and Dykman (2007) saw it as important to provide experience in both, along with a strong focus on techniques to encourage communication. Traditional processes rely heavily on documentation for communicating project methods and product goals among team members; this documentation is then used for “archaeological purposes” communicating product requirements and software design across years for product maintenance efforts (Kessler & Dykman, 2007, p. 313). Communication in Agile projects is more informal with direct interaction being preferred; documentation produced is just enough to ensure a maintainable product, with the main focus being to produce self-documenting code (Kessler & Dykman, 2007).

Kessler & Dykman (2007) defined a set of requirements for an application that was large enough to allow students to undertake a traditional process phase followed by an Agile phase. In the traditional phase the students undertook a waterfall approach of moving through the life-cycle phases of: analysis, design, implementation, testing, and software release; the analysis and design documentation the students were required to produce was

minimal so that the whole course time was not devoted to just requirements analysis (Kessler & Dykman, 2007). The agile phase involved use of Alistair Cockburn's Crystal Clear method utilising the Crystal Clear techniques of: methodology shaping workshop, blitz planning, daily stand-up meetings, side-by-side (pair) programming, and burn charts (which show development team progress) (Kessler & Dykman, 2007). Students asked to choose which approach they would use in their professional careers overwhelmingly selected a blend of the two processes; the suggested mix was 80% agile process and 20% traditional process, reflecting a preference to spend some time in design up front whilst still producing lightweight documentation (Kessler & Dykman, 2007).

North Carolina State University (NCSU) introduced students to some of the engineering practices in XP when looking to provide opportunities for students to collaborate and work in teams (Slaten, Droujkova, Berenson, Williams, & Layman, 2005). Slaten et al. (2005) describe traditional software engineering courses as having an emphasis on individual work, with collaboration viewed as cheating; when team projects are employed there is a "divide and conquer" approach, where the project is broken into small parts and individuals are assigned a part to work on independently (p. 323). The NCSU software engineering course was restructured so that students worked in pairs for the first assignment (following the waterfall method), individually for the second assessment (again following the waterfall method), and finished their semester by working in an agile team (Slaten et al., 2005).

The team project used the XP techniques of pair programming, user stories, short iterations, and acceptance tests (Slaten et al., 2005). A majority of the reflections on the experience from the participants were focused on the pair programming aspect, with students finding that the software they produced was of higher quality, due to the other person providing different solutions and helping find bugs, even though the communication of ideas between the pairs made them feel as though there was a decrease in productivity (Slaten et al., 2005). Slaten et al.(2005) note the students experienced an increase in confidence through the collaborative learning environment and the way in which the class experience emulated work place practices.

Rochester Institute of Technology utilised agile methods within their undergraduate software engineering course to address the challenge of teaching good software engineering practice in an engaging way whilst "providing a solid foundation for a student's life-long professional growth" (Reichlmayr, 2003, p. S2C 14). In this course the software teams applied the XP practices of: user stories, which introduced the students to simple project planning techniques and capturing user requirements; test driven development, which focused the attention of the students on the quality of the product early in the development cycle; and frequent delivery cycles, which gave the students an opportunity to reflect on and improve team processes (Reichlmayr, 2003). Interestingly, Reichlmayr (2003) chose not to enforce the use of pair programming within the software teams, even though this is one of the core practices within XP, stating that controlling the use of pair programming techniques was too

difficult as teams were often working outside of supervised lab times. Insisting on the use of this technique was described as a way of improving the course in the following semesters (Reichlmayr, 2003)

Sherrell and Robertson (2006) surveyed students who participated in an XP project within a University of Memphis graduate level software engineering course, describing previous research, that found that agile techniques and pair programming had resulted in improved code quality and a better learning experience, as mostly conjecture. Surveyed students found pair programming to be a positive experience, reporting that they felt the quality of their code had improved, as did their confidence and morale (Sherrell & Robertson, 2006) The user stories were reported as helping to deliver the correct product and students felt a greater sense of satisfaction and efficiency compared to when they had previously followed a traditional development approach (Sherrell & Robertson, 2006).

McDowell et al. (2006) compared data for different student intakes for an introductory programming course delivered at the University of California – Santa Cruz to try to determine if use of pair programming by novice developers would affect course completions rates and retention of students within the computer science major . Different intakes utilised different development approaches, with one intake being required to work individually on programming assignments, and the others undertaking pair programming to complete the assignments (McDowell et al., 2006). Along with the assignment code, students were required to submit a development time log, and were asked to rate how confident they were in their solutions and how much they enjoyed working on the assignment (McDowell et al., 2006). Results of the study showed that paired programmers had a higher completion rate for the course overall and when comparing the pass rates to those of the individual programmers it showed that the pairs were not just persisting due to the pressure of not letting down their partner, but actually were able to understand enough of the course material to pass (McDowell et al., 2006). The paired students were also more likely to take and pass a subsequent programming course, negating the theory that a weak student passes due to their partners' ability and then fails follow-on courses (McDowell et al., 2006). Paired students had greater confidence in the quality of their code and enjoyed working on the assignments more than the individual developers (McDowell et al., 2006).

Simon and Hanks (2008) interviewed students from two institutions, who had completed their introductory programming course utilising pair programming techniques, during their second programming course. Students reported trying out more ideas and potential solutions to programming problems when paired, as well as feeling they did not get stuck as often as when coding solo (Simon & Hanks, 2008). The pair programming techniques was beneficial to the students' learning and they felt as though it aided them in being successful in the course; drawbacks of the technique were to do with scheduling times to work on the assignment where both programmers were available and feeling as though they had accomplished less as the work was shared (Simon & Hanks, 2008).

Simon and Hanks (2008) make the following suggestions around implementing pair programming in an academic environment: reserve lab periods for students to work on pair programming assignments, rather than making them schedule time outside of class; promote understanding of the program code as a whole by having students explain the code as part of the assessment; provide students with a transitional assignment towards the end of the course where they code individually.

2.6 Real-world, team-based development projects: bridging the gap between theory and work readiness

Research suggests that the way to help prepare development students for the realities of their first job is to involve them in a project where: there is a real client or role play of a real client; the client requirements change or clients have conflicting priorities; students are made to work in teams; and the team works on an application with a large existing codebase (Begel & Simon, 2008a; Coppit, 2006; Dawson, 2000; Dawson et al., 1997; Hogan & Thomas, 2005; Joy, 2005). A team-based development project is a common element amongst many of the software engineering courses that have been mentioned, but few include all of these elements (Horning & Wortman, 1977; Kessler & Dykman, 2007; Leventhal & Mynatt, 1987; Ludi & Collofello, 2001; Postema et al., 2000; Reichlmayr, 2003; Shaw & Tomayko, 1991; Sherrell & Robertson, 2006; Slaten et al., 2005).

The importance of a team-based development experience for software developers is such that all suggested computer science and software engineering curricula include group work components and they are insisted upon by accrediting bodies, such as the British Computer Society and the IEEE (Joy, 2005). Sims-Knight and Upchurch (1998) state “the goal of education should be to produce effective practitioners” (p. 1304). Educators should aim to provide a learning environment which aids in not only learning declarative knowledge but provides opportunities for also developing procedural and metacognitive skills, allowing students to develop “habits of work that will sustain them through 10 years of deliberate practice” (Sims-Knight & Upchurch, 1998, p. 1304).

Providing experience of “programming-in-the-large”, as suggested by Shaw and Tomayko (1991, p. 40) in the large team project model, presents many challenges within the academic environment. Coppit (2006) describes the difficulties of implementing large projects in software engineering courses as being: there are additional management overheads for the instructor, students are not motivated by salary or benefits, instructors cannot fire non-performing students, students are not full-time in the course and have differing schedules which can impact on team progress, the project has a hard and fast deadline due to the academic end of semester, and students need to be graded individually not as a team. Coppit (2006) asserts it is necessary for the students to utilise state-of-the-art development tools and be involved in all of the different development activities so they gain experience in all of the facets of software development.

McMillian and Rajaprabhakaran (1999) describe three categories of experience that should be provided within the academic software engineering team project to help the students become work ready: technical foundations need to be laid through the use of specific tools and techniques; communication with multiple stakeholders (including team mates) needs to be practiced through written, oral and graphical (i.e. software design diagrams) methods; quality standards should be set so students learn to focus on reliability, maintainability, and correctness. Surveying 30 software professionals in management roles, McMillian and Rajaprabhakaran (1999) found that the most important element to include in academic software projects was working with real users. Respondents to the survey made many comments on the ill-preparedness of graduate developers to work with real clients, where their requirements are often ambiguous and subject to change (McMillan & Rajaprabhakaran, 1999).

Dawson et al. (1997) recommend that software engineering courses should contain a group project where the real world is simulated by: inadequate specifications so the students need to discover the requirements; utilising typical software practices such as, code inspections, review meetings, and preparing internal documentation; limiting development time to set hours; role play of key project stakeholders; role play customers with conflicting requirements; have a customer who is naïve about software capabilities and does not know what they want but then starts to contribute ideas as the project progresses; changing requirements, software, hardware and working conditions; sabotaging progress if no disasters have been experienced during the project; having a project review session to emphasise lessons learned. These real-life project simulations are designed to help develop the personal skills of communication, planning and adaptability required to work effectively in a team (Dawson, 2000).

Begel and Simon (2008b) advocate for a more realistic team-based software engineering project where students: debug a large existing codebase, write additional features, interact with more senior members to learn about the codebase; are given incomplete directions around requirements and testing and are left as a team to work them out. Brechner (2003) proposes that to address the difference between academic based projects and commercial ones there needs to be a course with a multidisciplinary (e.g., coders, designers, technical writers) project team that has real users, vague requirements, and strict deadlines; a large scale development course would address the need for graduates to be able to write code that will be integrated into a larger piece of software, developers need to be able to read, modify, and debug the code of others who are working on the same project.

To overcome the deficiencies graduates have in the areas of practical competence and social skills, as noted by industry professionals, Schlimmer, Fletcher, and Hermens (1994) proposed three main goals for computer science education: students learn about both theory and practice; students experience how to work with a team and others to efficiently create software; students learn how to create high-quality software, not just working programs. To

support these goals Washington State University implemented a software team practicum, where volunteers from a freshman programming paper were formed into a four-year experimental team to work on real world projects under the guidance of faculty and graduate mentors (Schlimmer et al., 1994). The practicum, a supplemental course not replacing any of the traditional courses on offer, was designed to provide students with experience using state-of-the-art tools and industry practices, as well as helping build teamwork and communication skills (Schlimmer et al., 1994). Industry partners were involved in the student evaluation, providing feedback on quality of the software developed, as well as presenting seminars on topical issues within the software profession; the long term nature of the software team practicum allowed for exposure to a broader range of software development problems, and students were able to form a clear understanding of the trade-offs and practices used in day to day software development (Schlimmer et al., 1994).

Moore and Potts (1994) implemented a three-quarter (one academic year) course practicum in Georgia Institute of Technology's Bachelor of Computer Science, which is undertaken during the student's third and fourth years. Dubbed the "Real World Lab", Moore and Potts (1994, p. 152) had three main pedagogical aims when designing the course sequence. Firstly, they wished to develop reflective learners who were able utilise good judgement to rigorously apply the correct techniques for any given situation; Secondly, they wished students to experience working within a team, aiding the development of communication skills required to work in a collaborative environment; Thirdly, they wanted to expose the students to working on on-going projects to remove the assumption that software development involves individual or group ownership of a product from conception to implementation (Moore & Potts, 1994).

The Real World Lab involves industry and provides the students with real problems, real clients, and real deadlines which they must meet (Moore & Potts, 1994). The benefits of the project experience are described as being: students are better placed to deal with complex problems and large systems, with the real client exposing them to the fact that requirements are never unambiguous and providing them with problems that are too big to be tackled individually; students learn autonomy and responsibility by having to manage their projects, stick within the schedule, and choose the appropriate tools and methods for the given problem; students become more flexible as they realise requirements change and are required to undertake maintenance on installed applications to ensure reliability (Moore & Potts, 1994).

To better mimic life, the Real World Lab requires new participants to submit a job application with an accompanying CV, students then move through a series of job levels (Moore & Potts, 1994). An entry level student is giving responsibility for a small task and is guided by senior students and academic staff when setting goals and tracking progress; during the second quarter of the course students are normally promoted to associate level (highly capable students may be promoted earlier), where they are assigned tasks which push their abilities

are given more responsibility to manage their own workload, often being made responsible to lead a small group on a subproject; final promotion is to principal level where they become student leaders responsible for client contact and managing and scheduling a project team (Moore & Potts, 1994). Student feedback on the experience showed that overall students valued the project experience feeling as though they had learned much from unpredictable nature of the real-life team-based project (Moore & Potts, 1994).

Inspired by the Real World Lab, Milwaukee School of Engineering introduced a three-quarter course, named The Software Development Laboratory, aimed at providing real world experience (Sebern, 2002). Software Development Laboratory is structured so many student teams work on large-scale ongoing projects, with the premise that inexperienced students work on well-defined areas of an existing application and more experienced students define requirements and architecture of new systems or new features (Sebern, 2002). As well as being part of a development team, students are part of "staff teams, such as the Software Engineering Process Group (SEPG), the Software Configuration Management group (SCM), the Software Quality Assurance team (SQA), the Planning and Tracking group (PT, and the Training Department (TD)" (Suri & Sebern, 2004, p. 22). The Software Development Laboratory works with real clients and student use an incremental development model, with each quarter initially containing two development cycles (Suri, 2007). Students are required to submit a process improvement proposal once a quarter, these are handled by the SEPG and Sebern (2002) credits these as being the most important driver of process improvement that has occurred within the lab. Improvements that have been made over the years include: restructuring the first quarter to only contain one development cycle to allow for several weeks orientation into the lab environment; improved planning and tracking, including the in-house development of a software tool; insisting that stakeholders are available for weekly communication with the students, with at least fortnightly face-to-face meetings (Suri, 2007).

Suri (2007) describes the main academic challenges around the Software Development Laboratory as being: the lack of time spent on development tasks by the average student (approximately 2.5 hours/week of the budgeted 10 hours/week) means there is a lack of progress on the software product; it is difficult to maintain student motivation and interest of real clients due to the slow progress of development; ratio of student teams to instructor means that there is a lack of mentoring; projects approved for the lab are non-critical due to the protracted development schedule, therefore client contact is often infrequent and busy stakeholders do not show much enthusiasm which is very de-motivating for the student teams; some projects require use of products and technologies instructors are not familiar with, meaning there is a lack of technical support for the students; students are unable to work out what their grades are based on the feedback they are getting during the course so they become frustrated. However, despite these challenges Suri (2007) asserts that students find the experience to be very valuable, with the ones involved in internships

between their junior and senior year commenting that the Software Development Laboratory experience does indeed reflect the real world.

Postema, Miller and Dick (2001) of Monash University introduced software maintenance into a second year software engineering practice course, stating that a majority of courses teach software engineering practice by involving student teams in developing new applications whilst it is more likely a majority of their professional life will be spent in maintenance programming activities. Erlikh (cited in Sommerville, 2007) states that maintenance accounts for up to 90% of all software costs. Students gain experience in the four types of maintenance activity: corrective, where errors are identified and fixed; perfective, where new functionality is introduced; preventative, where code is reengineered to make it more maintainable; adaptive, where software is changed to operate within a different environment (Postema et al., 2001). Postema et al. (2001) state student feedback shows the maintenance experience project as being a valuable one, providing an interesting way of extending students software engineering knowledge and preparing them for work.

The University of Kentucky took a real-world, team-based approach within their software engineering course to “provide hands-on experience” (Huffman Hayes, 2002, p. 192). The graduate level software engineering teams undertook the development of a phenylalanine milligram tracker application, designed to run on a personal digital assistant (PDA), which would aid sufferers of phenylketonuria disease in monitoring their diets (Huffman Hayes, 2002). Huffman Hayes (2002) reports the following benefits from the real-world project: teams were highly motivated to develop a quality product due to the serious nature of the problem; interaction with the user increased the teams commitment to successfully delivering the product; the students gained valuable experience in applying software reliability engineering principles.

Undergraduate software engineers at Georgetown University undertook a semester long simulation where the entire class formed a development team made up of four mini teams with formalised roles (Blake, 2003). The four teams were: the analysis team, the software design team, the development team, and the database team (Blake, 2003). The four teams worked together to design a solution to a local IT problem, sometimes involving a real client, otherwise the course instructor role played the client (Blake, 2003). Students within the teams were assigned specific roles, including a designated team leader who was responsible for keeping track of tasks and presenting to the other the teams the technical products developed during the week; teams were given secondary roles to ensure that their participation was maintained throughout the project life-cycle e.g., once the analysis was complete the analysis team were responsible for the design of test cases and the executing of test scripts (Blake, 2003).

Blake (2003) states the students learn valuable lessons around group collaboration, particularly during the transitional period where one teams responsibilities are about to end

and the teams are trying to hand over to the team responsible for the next phase of the life-cycle. Blake (2003) asserts that by the end of the course the students had the full picture of what is involved in the software life-cycle. However, the approach of assigning students to one particular phase of the life-cycle means the hands-on experience is limited to that role. Blake (2003) reports student feedback on the course was favourable with students stating it helped them understand the software life-cycle and better prepared them for their internships.

Queensland University of Technology has provided explicit support for the fostering of teamwork skills within their software engineering project courses, stating that many team-based software courses trust these skills will simply emerge from being part of a team (Hogan & Thomas, 2005). Hogan and Thomas (2005) found the iterative cycles of an Agile development method provided more opportunity to reflect on personal and team effectiveness, as well as providing the repetition needed for students to be able to learn and apply the team development processes. They developed an in-house Agile methodology, which they named Real World Software Process, to enforce the teaching of teamwork skills within their software engineering projects (Hogan & Thomas, 2005). The Real World Software Process provides the students with templates they must use for time management, meetings, and the actions that result from the meetings as well as guidance on their use; the methodology is focused on communication within the team rather than on individual productivity (Hogan & Thomas, 2005). The Real World Software Process is applied by students in a second year team project, by the end of which they have a clear understanding of the process and the importance of teamwork preparing them for the challenges involved in their final year team project involving an industry partner (Hogan & Thomas, 2005). The industry involvement provides the students with exposure to technologies and tools currently in use, and increases their motivation as they strive “to perform professionally in front of a potential employer” (Hogan & Thomas, 2005, p. 208).

Undergraduates at Iona College complete two capstone courses where they work in teams to become better prepared for their professional careers (Poger & Bailie, 2006). In the first course students perform analysis and create a design for a system that is built and implemented in the second course, to make the course more realistic and therefore more motivating for the students a real-world problem was introduced in 2002 (Poger & Bailie, 2006). Students were tasked with creation of a web based assessment system for the Computer Science department; the product was designed during the 2002-2003 academic year and implemented during the following year (Poger & Bailie, 2006). During design and implementation the students were expected to make changes and improvements to the system based on stakeholder feedback, with feature enhancements and on-going maintenance of the system still being completed by the students of these courses (Poger & Bailie, 2006). Employers of five of the students who had graduated after the initial design phase were asked to rate the graduates in the areas of: team work, interpersonal skills,

problem solving ability, oral and written communication skills; Four employers responded with positive feedback on the graduates, in particular their group skills and problem solving were rated highly (Poger & Bailie, 2006).

The Small Project Support Center was set up at Radford University to overcome the difficulties associated with on-going maintenance when student teams are involved in real-world development projects (Chase, Oakes, & Ramsey, 2007). The center was devised to: be a "clearinghouse" for projects, allow high level management of projects (e.g., coordinate the development of small parts of a project by different classes of students and then integrate the outcomes into a finished application, monitor and arrange client interactions), taking finished products and completing final testing before moving the project into production (i.e., turning it into a live system), and provide ongoing support for projects developed by students (Chase et al., 2007). The center has a full-time director but otherwise is staffed by part-time student workers (a mix of senior and juniors to ensure some experienced staff on the team), allowing the students to gain valuable experience during their studies and full-time summer work (Chase et al., 2007).

2.7 Conclusion

From the discussion above, it is clear that the best way to educate software developers has been debated for decades. The debate began with whether computing education was a science discipline or an engineering practice. Pranas (1999) described the need for both types of education depending on the students chosen career path. Those who see themselves as builders, designers of tools for others will follow the engineering path; those who wish to study things of interest to both groups to add to the body of knowledge will choose the scientific education. Even with the implementation of newly designed software engineering degrees, and a long history of computer science programs, comments from industry still suggest graduate developers are not prepared to work in the real world. This stems in part from the traditional model of education, and the traditional methods for educating software developers.

The growth of the computing industry was rapid and educators struggled to keep pace with the constant change in the new field. The demands of the new knowledge society meant organisations required graduates not only with specialist technical skills but also with excellent communication skills so they could effectively work in teams. Industry was reporting the need to run in-house training courses to bring new graduate employees up-to-speed with development practices as they happen in the real world. Educators responded to the cries from industry by implementing many evolving best practices into coursework. These included: measuring developer productivity using the Personal Software Process, bringing Agile development techniques into team-based project work allowing teams to adapt to user changes and work with peers to improve code quality, bringing modern development environments and tools that support collaborative development into coursework.

Research indicates that to best prepare software development students for real work they should be involved in a project where: there is a real client or role play of one, the client requirements change or are in conflict, students work in teams, and the team works on an application with a large existing codebase. The researcher implemented such a project as the learning environment described in this action research study.

Though there have been many institutes who have implemented the real-life team-based project approach, at the time of data collection the researcher was unable to find evidence in the literature that graduates of this approach have been formally surveyed once they were full-time employees to evaluate the usefulness of this approach. There is anecdotal evidence, obtained from students participating in internships post project, that the approach is realistic and a small sample of employers have indicated that the graduates are work ready with strong team work and problem solving skills. The researcher aims to fill this gap.

CHAPTER THREE

3.1 Introduction

This action research study used a convenience sample of students/graduates of the researcher's third year Software Engineering course and regional ICT employers. The researcher implemented a real-life, team-based software development project as the learning environment for the third year Software Engineering course taught within the Bachelor of Information and Communications Technology (Applied) at UCOL, Palmerston North, New Zealand. The purpose of this study was to validate and improve this learning environment. Graduates who participated in the learning environment were surveyed to determine the core skills they had achieved within this learning environment and to identify common gaps in their skill sets experienced as new employees. Their employers were also surveyed to gain their perspectives on what skills were of most importance in their graduate ICT workers. The researcher adapted the learning environment to address the identified gaps in the graduate skill set. The graduates of the second iteration of the real-life, team-based project were then surveyed and the learning environment was adapted again. In the third iteration of the real-life, team-based project, the researcher observed the students working within the learning environment and used these observations alongside student reflections to identify major challenges faced so the researcher could aid students to overcome these hurdles in future iterations of the real-life, team-based project.

3.2 Research title and significance of the study

Title: Evaluating the Effectiveness of a Real-Life, Team Based Software Development Project for Tertiary Students

Though there are a number of researchers that have identified inadequacies in traditional software engineering education and suggested strategies for providing students with a more realistic learning environment there appears to be have been little evaluation of how the real-life project experiences and the skills gained from them have been viewed by graduates once they are in the workforce.

This research aimed to validate and improve the learning environment provided to the software engineering students on the Bachelor of Information and Communications Technology (Applied) degree with the goal of making graduate transition to the workplace more seamless. It aimed to confirm that the skills being taught within the learning environment would make the graduates more employable and also address the requirements of the regional ICT community. The information on the learning environment and techniques used to help students overcome difficulties faced in the classroom will add to the body of knowledge on tertiary teaching methods, particularly in relation to technology-based courses.

The research on the effectiveness of the learning environment may be useful to other institutions in New Zealand looking to implement a similar approach.

3.3 Research questions

The study aimed to answer the following research questions:

1. Do non-technical skills (e.g., communication, problem solving and decision making, self-management, and teamwork) continue to be of high importance to employers hiring ICT graduates?
2. Are the development languages, technologies, and processes utilised in the project still currently in demand in local industry?
3. What skills did the graduates perceive as improved or formed by participating in the real-life development project?
4. Of the skills demonstrated during the project which ones were required of the graduate developers in their first ICT role?
5. Did the learning environment provide real-world experience that the novice developers could use as evidence of skills during employment interviews?
6. Are there common themes amongst the skills the graduates identify as lacking in their knowledge base that could be addressed by adaptation of the learning environment?

3.4 Research design

The researcher chose an interpretivist approach to answer the research questions, using action research as the methodology. "Interpretive paradigms strive to understand and interpret the world in terms of its actors" (Cohen, Manion, & Morrison, 2000)

The origin of action-research is generally attributed to Kurt Lewin, who in a 1946 article on minority groups stated:

> "The research needed for social practice can best be characterized as research for social management or social engineering. It is a type of action-research, a comparative research on the conditions and effects of various forms of social action, and research leading to social action. Research that produces nothing but books will not suffice" (Lewin, 1946, p. 35)

Lewin (1946) described the process of action research as following a recurring 'spiral' of steps of planning, acting, and reflecting on the result of the action (p. 38).

Reason and Bradbury (as cited in Stringer, 2010) define action research as:

> [Action Research] seeks to bring together action and reflection, theory and practice, in participation with others, in the pursuit of practical solutions to issues of pressing concern to people, and more generally the flourishing of individual persons and their communities (p. 313)

Action research is an iterative process in which the researcher endeavours to improve a practice, or outcomes of that practice by: investigating the problem, adapting a practice, and reflecting on the outcomes. Stringer (2008) states that teachers must "engage in a systematic process of inquiry and planning" to meet the needs of their learners with action research providing the framework to accomplish this task (p.15). The use of action research in education became more prevalent in the literature from the 1990s (Stringer, 2010). Carr and Kemmis (2009) identified three different types of action research based on the purpose of the project: technical, practical, and critical. In technical action research the participant-researcher aims to improve the outcomes of a practice; practical action research takes into account the voice of others with the aim of the participant-researcher acting more 'wisely and prudently' so the long-term outcomes of a practice will be better; critical action research is a collaborative process in which the participants "aim to change their social world collectively, by thinking about it differently, acting differently, and relating to one another differently" (Kemmis, 2009, p. 470). This research project fits into the practical action research area; with the researcher obtaining the opinions of both graduates and employers to improve the long-term outcomes (i.e. employability and seamless transition from student to graduate developer) for students undertaking the third year software engineering course.

3.5 The learning environment described

The learning environment was a specialist computer laboratory situated in UCOL's Palmerston North campus. The room contained 30 desktop computers running the windows operating system. UCOL has a Microsoft campus license meaning a majority of the software deployed to the UCOL leased desktops is from this organisation. The PCs in the specialist computer laboratory are under the control of the researcher, allowing the deployment of updated versions of software, and also open-source software solutions, as necessary without having to conform to organisational timeframes for software deployment.

As the real-life project idea was to design, develop and implement a software solution for the Medical Imaging Technology (radiography) students at UCOL, the development software utilised needed to be compatible with UCOL being a Microsoft Campus. In the first few weeks of semester one, 2008, a feasibility study was undertaken by the researcher and students of the third year software engineering class to determine the type of software solution that would be most suitable for the Medical Imaging Technology students and lecturers. The outcome of the feasibility study was the recommendation that a mobile application is developed for student use in clinical placement and a supporting web

application is developed for clinical supervisor use, allowing them to monitor student progress during placement.

The following software was utilised in the learning environment to facilitate this development: Windows XP, Microsoft Visual Studio 2005, and Microsoft SQL Server 2000. The development language chosen was C# with the .NET compact framework for the mobile application, ASP.NET for the web application, and Transact-SQL for the database. The mobile application was designed to run on Windows Mobile 5.0, which at the time was the current mobile operating system deployed on devices such as personal digital assistants (PDA). The development of the mobile application using the .NET compact framework and transferring of the data between the mobile device and the corporate SQL Server database were new to both students and researcher alike, this led to a very collegial learning experience, with a lot of trial and error, as we went through the steps of setting up a test network to prove the technology would work within the UCOL IT environment.

3.6 Sampling and Distribution

This action research study used a convenience sample. Enrolled students (Semeser One, 2011) and graduates (2008, 2009) of UCOL's Bachelor of Information and Communications Technology third year Software Engineering course (28) and a selection of regional IT employers (8) were the research participants in this study. Graduates who had been employed fulltime in software developer roles prior to studying were not included in the sample as they would not be classed as novice developers. The regional IT employers were selected from organisations who have employed the graduates (e.g., Provoke Solutions, Ministry of Culture and Heritage, Core Technology) and organisations who are members of UCOL's Bachelor of Information and Communications Technology industry advisory committee (Fujitsu NZ, NZ Pharmaceuticals, Advantage Computers). As all participants being surveyed were either ICT students or ICT professionals the least invasive manner of obtaining information was by electronic distribution of the survey. The participants were invited via email to complete an online survey at their convenience.

3.7 Instruments

The researcher used different data collection techniques during the research project. In the first and second iteration of the project, graduates were surveyed in order to determine the degree of skills developed within the learning environment and to try and identify common gaps in their skill sets that they experienced as new employees. For each of the different skills demonstrated within the learning environment, graduates were asked to rate how much they learnt of that skill in the project and how useful it has been to them so far in their career, this is an adoption of a scale applied by Lethbridge (1998b) in his survey of software professionals to gauge their perception of the relevance of their education. Graduates were asked to provide their employer details, once they had checked with their managers that they

would be open to receiving information about the research project. The employers were surveyed to find what they thought were the core skills required by graduate developers. In the third iteration of the project, the researcher used observations alongside reflective journals submitted by the students to identify the major challenges faced by the students in the learning environment to develop strategies to assist the students to overcome these hurdles in future iterations of the learning environment. Whilst many researchers would call this 'triangulation' of the data, Bogdan and Biklen (2003) warn against use of this term in qualitative research stating "it confuses more than it clarifies, intimidates more than it enlightens" (p.107).

3.7.1 Graduate Surveys

Questions differed for the two groups of graduates as the skills utilised, the structure of the development teams, and the learning environment differed for each group.

Graduates of the first iteration of the learning environment were in two distinct groups: those who worked on the mobile application and those who worked on the web-based application and database. The learning environment was constructed in this first iteration (see 3.5 The learning environment described).

Graduates of the second iteration of the adapted learning environment (a smaller cohort) participated in both the mobile application and web application, maintaining the software and also developing new features.

Development skills used differed for each group; soft skills were similar for both iterations.

For each of the different skills demonstrated within the learning environment, graduates were asked to rate how much they learnt of that skill in the project and how useful it had been to them so far in their career (refer Appendices One and Two).

3.7.2 Employer Surveys

The employers of the graduates were surveyed (Refer Appendix Three) to identify the core skills required of their new ICT graduates. Questions for this survey were created using categories adapted from previous local studies (Bekesi & Gardner, 2003; Snell et al., 2002; Young et al., 1999) and skill surveys conducted by international recruitment agencies and IT consultants (Argent, 2006; Watson, 2010). The employer survey was distributed to colleagues of the researcher, who had in-depth IT knowledge, for validation. This led to the inclusion of two additional categories that were considered to be of importance within their fields: server and desktop virtualisation, storage area networks and perserverance.

3.7.3 Observations and Reflective Journals

In the third iteration of the learning environment the researcher was a participant-observer during classroom observations of the adapted learning environment. Observations were recorded in an electronic format at times when the researcher was not required for role-play. These observations were used, alongside reflective journals submitted by the students, to identify challenges with use of the learning environment to enable the researcher to identify strategies to further aid students in future iterations of the learning environment.

3.8 Data collection and analysis

3.8.1 Procedures and instruments

Preliminary Activities

A literature review was undertaken. This included an examination of the previous software engineering class that was involved in the real-life team-based project to identify the participants and document the learning environment activities and software resources used.

Data Collection Phase One

Graduates, who gained employment in developer roles, were surveyed in order to determine the skills developed within this learning environment and to try and identify common gaps in their skill sets experienced as new employees. Graduates of the first iteration of the learning environment were in two distinct groups: those that worked on the mobile application and those that worked on the web based application and database. For each of the different skills demonstrated within the learning environment, graduates were asked to rate how much they learnt of that skill in the project and how useful it had been to them so far in their career. This is an adoption of a scale applied by Lethbridge (1998b) in his survey of software professionals to gauge their perception of the relevance of their education.

Graduates obtained permission to provide the researcher with their employer's email address. Employers were invited by email to complete an online survey to identify the core skills required of their new ICT graduates.

Phase One Data Analysis and Learning Environment Adaption

Qualitative analysis (refer section 3.8.3 for coding frame) of the questionnaires yielded data for:

- evaluating the usefulness of the learning environment at preparing the graduates for work; and
- helping identify common gaps in skill sets.

This was followed by an investigation into adaptation of the learning environment to address the identified skill gaps, followed by implementation of viable changes.

Data Collection Phase Two

Graduates, from the second iteration of the real-life, team-based project who gained employment in developer roles, were surveyed to review the effectiveness of the adapted environment. This survey aimed to validate the relevance of the skills developed within the adapted learning environment and to identify if there were still gaps in their skill sets after the redesign. Graduates obtained permission to provide the researcher with their employer's email address. Employers were invited by email to complete an online survey to identify the core skills required of their new ICT graduates.

Phase Two Data Analysis and Learning Environment Adaption

Data analysis performed was the same as phase one, this led to further adaption of the learning environment to address identified skill gaps.

Data Collection Phase Three

The final stage of the research project involved classroom observations of students working within the adapted environment. Students who participated in this iteration of the real-life team-based project were required to complete reflective journals documenting their development experience. These journals were used, alongside the classroom observations, to gain an insight into major difficulties experienced by the students working within the adapted environment to allow the researcher to aid students to overcome these hurdles in future iterations of the real-life, team-based project.

The researcher was a participant-observer during classroom observations of the adapted learning environment. The researcher role-played the part of the client and also acted in a mentoring role, like a senior software developer would with new graduates in a work situation. Observations were recorded in an electronic journal during the times the students did not require the researcher for role-play or mentoring so there was no disruption to the normal classroom activity

3.8.2 Administration

Data collected were stored electronically while analyses were carried out. All files were stored on a password-protected laptop in the researcher's office at UCOL and transferred to the research supervisor's office at SMEC at Curtin University at the end of the study and will be kept for five years after which they will be destroyed.

3.8.3 Data analysis

The closed question responses from the graduate surveys were coded into numeric form. This was done to allow the researcher to quickly ascertain whether a particular skill had been useful to a majority or only a few of the graduate participants. The open-ended question on skill gaps from the graduate surveys were categorised using the following coding frame:

1. Team-based development tools
2. Database related
3. Team communication skills
4. Syntax related

The observations and reflective journals were categorised using the following coding frame:

1. Team-based development environment
2. SCRUM / Agile Development methods
3. Orientation of large existing code base
4. Team conflict
5. Time management

Strategies devised to help future students overcome the common difficulties experienced within the learning environment were described by the researcher.

3.9 Limitations

Due to small class sizes for two of the three years that this research was conducted the survey sample of graduates is small (eight responses from a possible eleven graduates). This also led to a small employer sample (eight responses).

3.10 Ethical considerations

The researcher had a teacher-student relationship with some of the participants. Graduate participants were surveyed once they had obtained their first development role on graduating, therefore there was no dependency on the researcher when the participants were providing their views and opinions on the classroom activity and the relevance of the skills obtained during the semester. Consent was gained from the currently enrolled students who underwent observation in a session at the beginning of the semseter. The researcher verbally described the research project and gave participants the opportunity to ask questions. Participants were reassured that they had the right to withdraw from the research at any time, without prejudice or negative consequences and that no aspect of the

research will be used in determining their final grade in the course. The researcher left the room and the participants were then provided with an information sheet that clearly stated the purpose of the intended research and consent form by an independent party so they did not feel pressured to provide consent. There was minimal disruption to the normal state of affairs during the classroom observation. Pseudonyms were used throughout (through referring to name of group only), and some information was changed to guarantee the participants anonymity. As all participants being surveyed were either ICT students or ICT professionals the least invasive manner of obtaining information was by electronic distribution of the survey. Refer Appendix Four for Information and Consent forms.

3.11 Summary

This action research study went through multiple iterations and utilised multiple data collection methods. Graduates were surveyed to obtain their view on how useful particular skills developed in the learning environment have been to them in their career; employers were surveyed to obtain their opinion on the core skills required from their ICT graduates. Skill gaps identified by graduates, and from secondary data sources, were addressed by adaption of the learning environment. The researcher's observations in the third iteration of the learning environment were used, alongside student's reflective journals, to identify themes related to difficulties experienced with using the learning environment. The researcher used these themes to devise strategies which can be implemented in future iterations of the learning environment to make the experience smoother for the participants.

CHAPTER FOUR

4.1 Introduction

Data for this action research study were obtained from: two groups of graduates of the real-life, team-based project who gained employment in developer roles; a sample of regional employers, made up of employers of the graduates and members of UCOL's BICT industry advisory committee; classroom observations of students utilising the re-designed learning environment; and the students' reflective journals. An overview of the aims of the graduate survey is provided, and then the responses from the two graduate samples are presented. This is followed by a summary of issues encountered when using the re-designed learning environment, as mentioned in the reflective journals and/or noted by the researcher during classroom observation. Finally, responses from the regional employer survey are presented.

4.2 Graduate Survey Overview

Graduates, from two iterations of the real-life, team-based project who gained employment in developer roles, were surveyed to review the effectiveness of the learning environment. These surveys aimed to validate the relevance of the skills developed within the learning environment and to identify gaps in the graduates' skill sets. The graduate surveys were designed to determine the answers to the following research questions:

- What skills did the graduates perceive as improved or formed by participating in the real-life development project?
- Of the skills demonstrated during the project, which ones were required of the graduate developers in their first ICT role
- Did the learning environment provide real-world experience that the novice developers could use as evidence of team-based skills during employment interviews?
- Are there common themes amongst the skills the graduates identify as lacking in their knowledge base that could be addressed by adaptation of the learning environment?

The surveys were divided into two sections. In iteration one, section one contained questions related to team-based development and real-life clients and section two focused on technical skills that were developed. In iteration two, section one contained questions related to team-based communication and team development tools and section two focused on the technical skills developed. For each of the different skills demonstrated within the learning environment, graduates were asked to rate how much they learnt of that skill in the project and how useful it had been to them so far in their career. This is an adoption of a scale

applied by Lethbridge (1998b) in his survey of software professionals to gauge their perception of the relevance of their education.

4.3 Graduate Survey Iteration One

The development students were split into two distinct teams during the first iteration of the project: those that worked on the mobile application; and those that worked on building the database and constructing the ASP.NET web application. The total class size was seven, with two developers working with the researcher on developing the mobile (PDA) application and five developers working on the database and web application. From this sample a response rate of 71% was received, two mobile developers and three web developers (refer Figure 4.1).

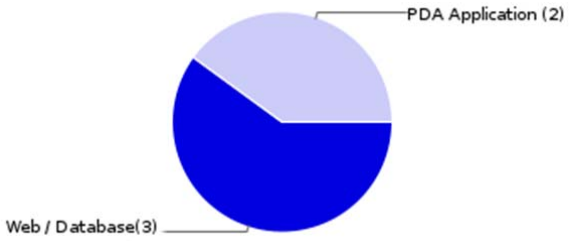


Figure 4.1: Iteration One - response rate by development activity

4.3.1 Learning Environment Overview

The learning environment was a specialist computer laboratory situated in UCOL's Palmerston North campus. The room contained 30 desktop computers running the Windows operating system version XP. UCOL has a Microsoft campus license meaning a majority of the software deployed to the UCOL leased desktops is from this organisation. In the first four weeks of the project, a feasibility study was undertaken by the researcher and students of the third year software engineering course to determine the type of software solution that would be most suitable for the Medical Imaging Technology students and lecturers. The outcome of the feasibility study was the recommendation that a mobile application be developed for student use in clinical placement and a supporting web application be developed for clinical supervisor use, allowing them to monitor student progress during placement.

As the real-life project idea was to design, develop and implement a software solution for the Medical Imaging Technology (radiography) students at UCOL, the development software used needed to be compatible with UCOL being a Microsoft Campus. The following software was utilised: Windows XP, Microsoft Visual Studio 2005, and Microsoft SQL Server 2000 running on Windows Server 2003. The development languages chosen were: C# with the .NET compact framework for the mobile application; ASP.NET for the web application; and Transact-SQL for the database transactions. The mobile application was designed to

run on Windows Mobile 5.0 which, at the time, was the windows mobile operating system deployed on devices such as personal digital assistants (PDA). The development of a mobile application using the .NET compact framework and transferring of the data between the mobile device and the corporate SQL Server database using Remote Data Access (RDA) was new to both students and researcher alike. This led to a very collegial learning experience, with a lot of trial and error, as the group went through the steps of setting up a test network to prove the technology would work within the UCOL environment.

4.3.2 Section One – Team-Based Development and Real-life Client

The first section of the graduate survey contained questions related to the team-based development skills and working with a real-life client. The skills were categorised under the following areas:

- Agile project management using SCRUM methodology;
- Peer programming techniques;
- Requirements elicitation with real clients;
- Team dynamics.

For each of these areas graduates rated how much they learnt about that skill during the project (Learned in depth (expert), Learned a lot, Became Functional (moderate working knowledge), Learned the basics, Became vaguely familiar), and how useful the skill has been to them so far in their development career (Essential, Very useful, Moderately useful, Occasionally useful, Almost never useful).

The first skill the graduates were asked to rate was use of Agile project management techniques. The students concentrated on two of the key aspects of the Agile method SCRUM: prioritised product backlogs and daily stand-ups. Product backlogs are a list of software requirements which are prioritised by the product owner (person who represents the organisation that initiated the software development project). The top priority features on the product backlog are developed first, thus if the project runs out of time or reaches budget limits the most important requirements will have already been addressed. Daily stand-ups are micro-meetings that kick off the development session. During the stand-up each developer responds to three questions:

- What have you done since the last stand-up,
- What are you working on now,
- Is there anything standing in your way of getting the work done.

The meetings are conducted standing up to ensure brevity. The aim is to make sure everyone on the project team knows what is going on and that there is no external interference to team progress. Whilst all of the graduates responded they had learned a lot

or became functional with these techniques, only three of them reported them very useful in their graduate roles (refer Figure 4.3).

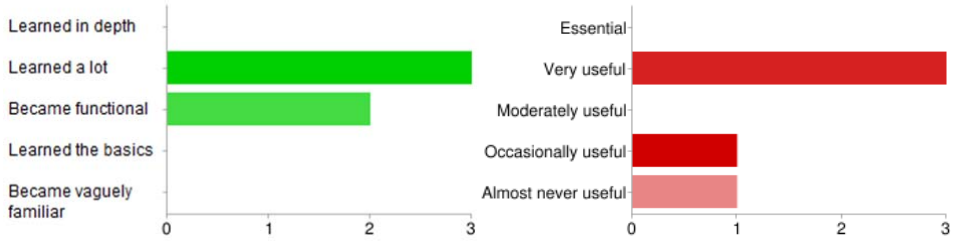


Figure 4.3: Agile Project Management using SCRUM methodology

Within the Agile methods there is a development technique known as peer-programming. When using this technique two developers sit side-by-side, one is the driver (in control of entering the code) and the other is the navigator. The role of the navigator is to assist with problem solving and code quality. All of the graduates became functional or learned a lot about this technique during the project. A majority reported it useful or very useful in their career with only one graduate responding as only using it occasionally (refer Figure 4.4).

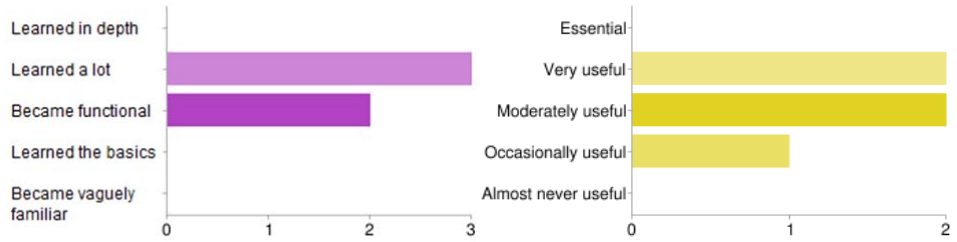


Figure 4.4: Peer Programming techniques

One of the graduates made further comment regarding this technique:

“In my work place peer programming is vital in facilitating efficient and robust modules”

In iteration one, the students were involved in a number of activities that interacted with project stakeholders other than the researcher. During the feasibility study they ran focus groups with Medical Imaging Technology students and lecturing staff to determine the core application requirements. Once technology choices had been made they presented the outcome to both the lecturer users and IT staff, in order to obtain approval on user requirements and application infrastructure before moving into the development phase of the project. Once a working application was ready, user training was conducted and feedback on use of the device was recorded so modifications could be made. A training session was also delivered to clinical staff at the hospital prior to the first pilot test of the mobile application. Four of the five graduates responded as having learned a lot or becoming functional with the client facing techniques. Three of the five graduates reported these skills to be essential or very useful (refer Figure 4.5).

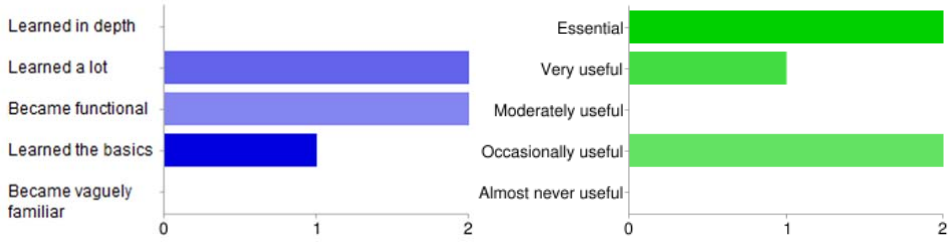


Figure 4.5: Requirements Elicitation with real client

4.3.3 Section Two – Technical Skills

Section two of the survey contained questions related to the technical skills that were developed during the project. Graduates answered different questions based on whether they were part of the web and database development team or the mobile application development team. For each of the technical skills, graduates again rated how much they learnt about that skill during the project and how useful the skill has been to them so far in their career.

4.3.3.1 Technical Skills - Web Application and Database

The web and database skills were categorised under the following areas:

- ASP.Net web development,
- SQL Server database management system,
- Web application security.

All of the respondents from the web development team either learned a lot or became functional with the ASP.Net web development techniques. Only one of the graduates reported it as being very useful, with the two other respondents reporting they never use or only occasionally use these skills (refer Figure 4.6).

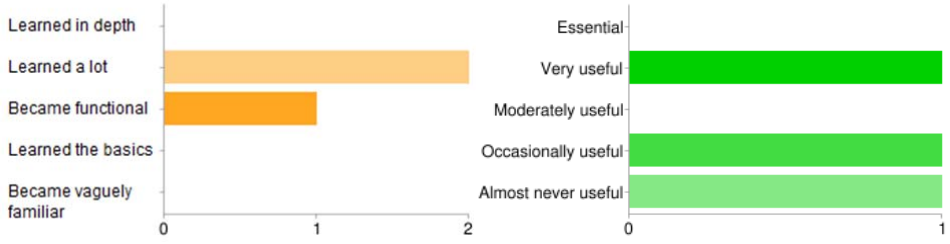


Figure 4.6: ASP.Net Web Development

Utilising the SQL Server database management system (using Transact-SQL query language) was the most useful of the technical skills for the web / database developers with all three graduates responding as very useful. All respondents became functional or learned a lot about these skills during the project (refer Figure 4.7).

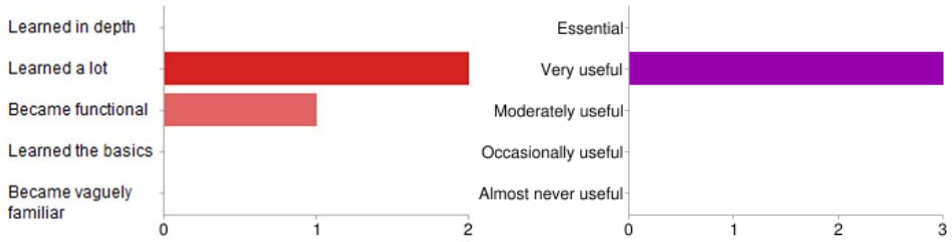


Figure 4.7: SQL Server Database Management System

The database skills were the most transferrable, meaning that skills learnt within this particular database management system could be adapted and applied to other relational databases.

The skill that the web development / database team learnt the least about was web application security, with two reporting they learned the basics and one respondent saying they had become functional (refer Figure 4.8). There was less emphasis on the security of the web application as it was going to be hosted on the UCOL intranet (a site only available internally). Users would access the application via the 'single sign-on' feature available under Windows domains (i.e. a user logged into the Windows domain has their credentials passed over to other windows applications residing within the domain). One of the graduates reported these security skills to be moderately useful, with the other two respondents stating they were occasionally useful (refer Figure 4.8).

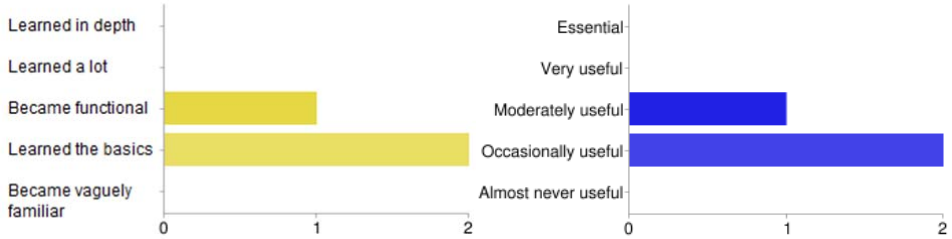


Figure 4.8: Web Application Security

4.3.3.2 Technical Skills - Mobile Application

The mobile application developer skills were:

- C#.Net and the .Net compact framework,
- RDA data transfer with compact edition database,
- Mobile application security (e.g. privileges, user authorisation, and data encryption).

C#.Net using the .Net compact framework was rated as being essential by one developer and very useful by the other. Though the .Net compact framework is used only with Windows mobile applications, the C#.Net skills would be transferable to other .Net application development projects. One graduate felt they had learned these techniques in depth, the other responded as having learnt the basics (refer Figure 4.9).

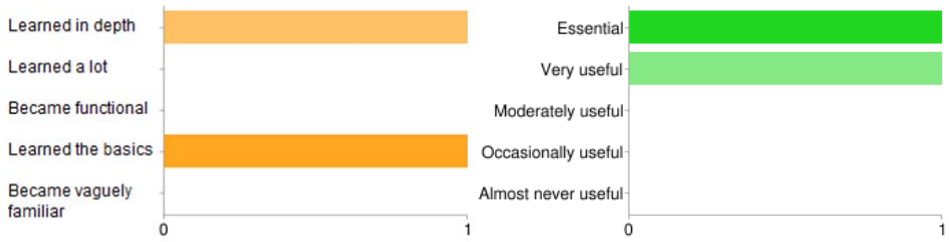


Figure 4.9: C #. Net and the .Net Compact Framework

RDA (remote data access) is a data transfer technique that is used by mobile applications to push and pull data records from the main database. These skills are again proprietary to Windows mobile applications. Though the syntax learnt would not be a transferable skill, the knowledge behind how the data are transferred from the mobile application to the corporate database would be something that could be applied to different IT infrastructures the graduates were faced with in the future. One graduate felt they had learned a lot about the technique and had found it very useful, the other responded as becoming functional and using the skill occasionally (refer Figure 4.10).

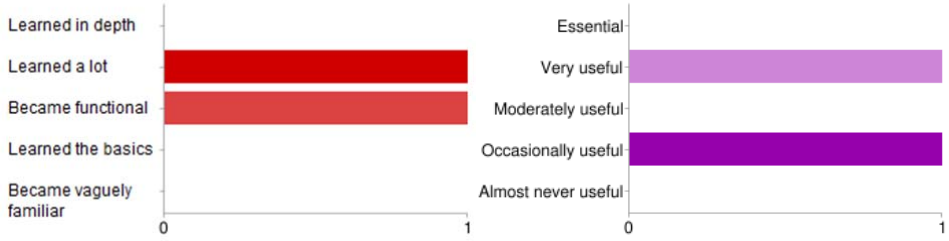


Figure 4.10: RDA data transfer and the compact edition database

There was a large emphasis on data encryption and security for the mobile application. The UCOL network consultant specified everything be encrypted as the application was to access a UCOL hosted database. Both mobile developers felt they had learned a lot about mobile application security, one graduate had found these skills to be essential and the other graduate found them to be very useful (refer Figure 4.11). The concepts of data encryption would be transferable to other application development environments. Further comment on the technical skills developed was made by one of the mobile application developers:

The MIT project gave me experience in the field of IT that I have chosen to work in

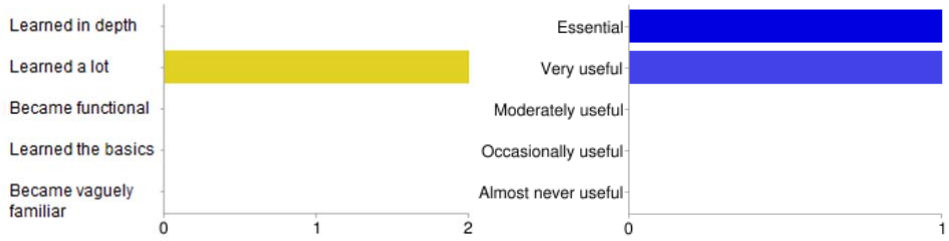


Figure 4.11: Mobile Application Security

4.3.4 Real-life project experience

The final section of questions related to the real-life project experience. All of the graduates felt their confidence as a developer had improved due to the experience (refer Figure 4.12).

Figure 4.12: Improved confidence as a developer

It is the researcher's experience that graduates who are technically competent often doubt their abilities until they are put to the test in a 'real job'. Additional comments provided by the graduates validated the confidence building aspects of the learning environment. Sample comments from two of the respondents were:

> *I found that whilst doing the MIT project, fundamental skills were learnt that helped me understand Software Development in the real world*
>
> *The knowledge learnt from the MIT project aided me in helping to change the way my first employer developed software*

All graduates used the project as evidence of experience on their curriculum vitae (refer Figure 4.13) and four out of the five graduates referred to the project experience in their job interviews (refer Figure 4.14).

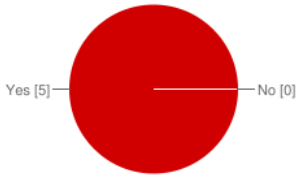


Figure 4.13: Project used as evidence of experience on CV

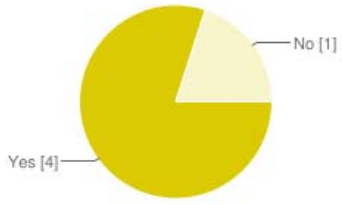


Figure 4.14: Project used as evidence of experience in interview

A majority of the graduates who referred to the project in their interview used it as an example of experience in a particular technology (refer Figure 4.15).

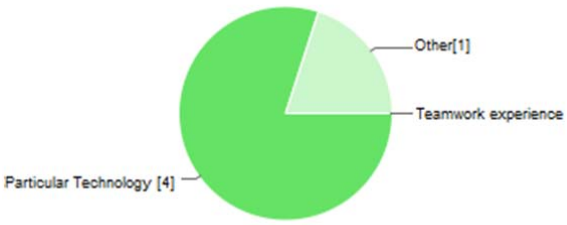


Figure 4.15: Aspect of project referred to in interview

In relation to skills developed one graduate comment is of particular interest:

> *the technical skills which were learnt did contribute towards my portfolio but it was those unrealised skills which played the crucial role, which I didn't realise until I was being interviewed in the industry world and started my job*

These "unrealised skills" were the team-related soft skills, which are often considered unimportant by technically-oriented graduates.

One graduate made extensive further comment regarding his interview. He was asked how he would cope in the position when he had no previous experience with the particular software development environment he would be working with. His response:

> *It is just a matter of learning syntax and the ambiguity of learning different software is not new to me. I guess if it wasn't for MIT [project] I wouldn't have been able to answer that question with that confidence. I was glad that there was a real example I could provide which added to my credits*

4.3.5 Skill Gaps

Graduates were asked if there were any skills that were required of them when they began their first ICT role that could be learnt within the classroom. The responses were coded using the following categories:

- team-based development tools,
- database related,
- syntax related,
- team communication skills (refer Figure 4.16).

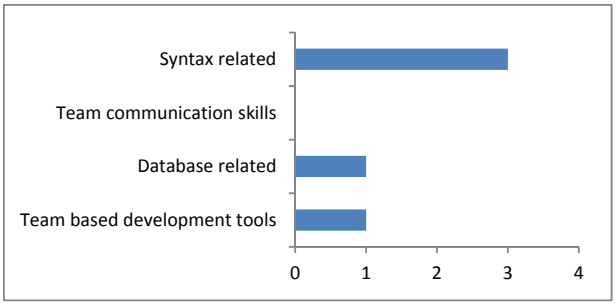


Figure 4.16: Skill Gaps

Three graduates mentioned web related skills and another responded with a database related skill that had already been introduced into other courses in the BICT curriculum (JavaScript, JQuery, LINQ, XSLT and XML). The final skill gap identified was source control, a team-based development tool that allows multiple developers to work on the same code in a controlled manner (i.e. any conflicts with changes made by different developers is notified by the software so they can be resolved). The real-life, team-based project was the ideal place to address this skill.

4.3.6 The redesigned learning environment

Based on the identified skill gap, the researcher selected Microsoft Visual Source Safe for source control. At the time, this was the recommended source control solution when using the Visual Studio development environment. Other changes to the learning environment were updates due to technology changes.

A new version of Visual Studio (2008) was implemented to replace the 2005 edition. There were also updated versions of the Windows mobile operating system and the compact edition database, so Mobile 6.0 and Compact edition 3.1 were added to the updated Visual Studio 2008.

4.4 Graduate Survey Iteration Two

The cohort of students in the second iteration was again small, with only five enrolled. Of the five, four were invited to participate in the graduate survey. The student who was omitted from the sample had previous software development experience in industry. Of the four invited, three of the graduates responded and completed the survey.

4.4.1 Learning Environment Overview

The learning environment was a specialist computer laboratory situated in UCOL's Palmerston North campus. The room contained 30 desktop computers running the Windows XP operating system. Changes made to the learning environment based on feedback from past graduates, and due to technology updates, meant the following software was utilised in the second iteration of the project: Windows XP, Microsoft Visual Studio 2008, Windows

Mobile 6.0 using Compact edition database 3.1, Microsoft Visual Source Safe 2005, and Microsoft SQL Server 2005 running on Windows Server 2003. The development languages were: C# with the .NET compact framework V2 for the mobile application; ASP.NET for the web application; and Transact-SQL for the database transactions.

The other key element that changed for the second iteration of the project was that there was now an application with a large existing code base that had undergone a user pilot test. Students would have the experience of orientating themselves in the existing application infrastructure before they would be ready to make any changes to the application in response to issues that were raised by users during the initial pilot test. There were also not enough students to utilise peer-programming techniques, so the researcher opted for putting in place a discussion forum for team-based communication rather than focusing on agile development techniques and SCRUM. All students in this cohort were involved in maintenance and update of both the mobile application and web application.

4.4.2 Section One – Team-Based Development: Communication & Tools

The first section of the graduate survey for iteration two contained questions related to the use of team-based development tools and communication within the team. The skills were categorised under the following areas:

- Code repository and version control;
- Discussion forums for team communication;
- Orientation in a large existing codebase;
- Team dynamics

For each of these areas graduates rated how much they learnt about that skill during the project (Learned in depth (expert), Learned a lot, Became Functional (moderate working knowledge), Learned the basics, Became vaguely familiar), and how useful the skill has been to them so far in their development career (Essential, Very useful, Moderately useful, Occasionally useful, Almost never useful).

The software Visual Source Safe provides a central code repository and simple version control that notifies if conflicting versions of the code are being ‘checked-in’. One of the graduates felt they had learned a lot about source control, one felt they had learned the basics and one responded they had become vaguely familiar. Two of the graduates reported these skills to be very useful or essential, with only one responding that they were only used occasionally (refer Figure 4.17).

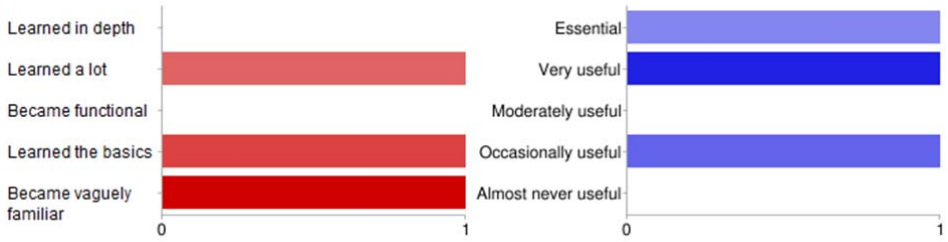


Figure 4.17: Code Repository and Version Control (Visual Source Safe)

The participants in this iteration worked on separate parts of the application simultaneously. It was agreed by the group that they would restructure the mobile application so it was more maintainable in the future. The main aim of the restructuring was to separate the code that delivered the user interface from the code that contained the logic to process and store the data. This initial process of restructuring the application required a high level of communication among the group as all of their individual parts were going to be affected. The student who worked on the restructuring thoroughly documented the process within the discussion forum so other participants could alter their parts of the application to fit the new structure. Two of the graduates felt they had learned a lot about use of discussion forums for team communication, one responded as having learnt the basics. Two graduates reported this skill to be essential, one felt that it had been moderately useful (refer Figure 4.18).

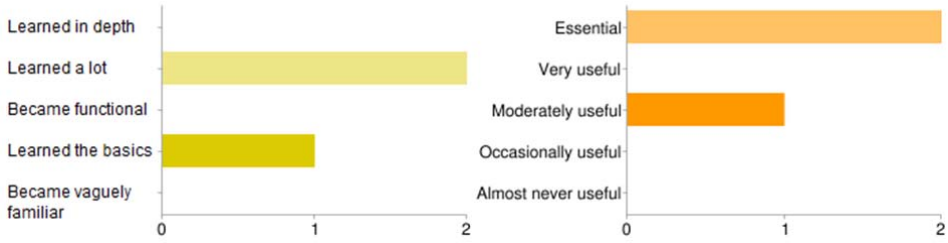


Figure 4.18: Team-based communication using discussion forums

Graduates were asked to rate how much they learnt about team dynamics, examples provided were: communication, priority negotiation, and conflict resolution amongst team members. Two responded they had learned a lot and one that they had become functional. One graduate reported these skills to be essential, the other two responded they were very useful (refer Figure 4.19).

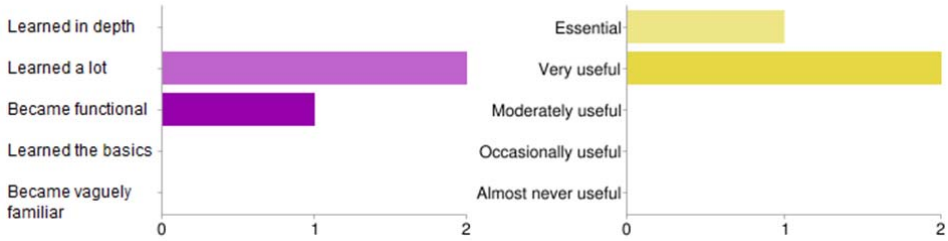


Figure 4.19: Team Dynamics

The final aspect of the non-technical skills was orientation within a large existing codebase (getting your head around code someone else has written). Two graduates learned a lot and one became functional with these orientation skills. All graduates reported these skills to either be essential or very useful (refer Figure 4.20).

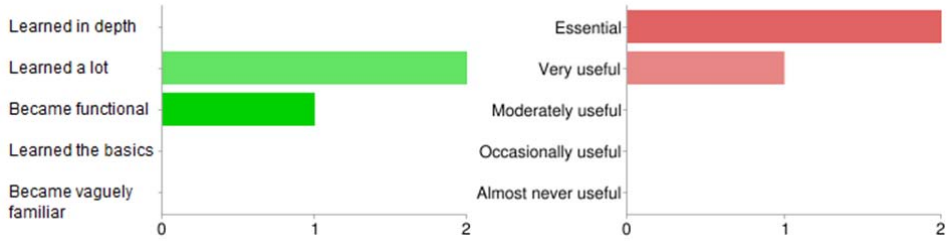


Figure 4.20: Working with a large existing codebase

4.4.3 Section Two – Technical Skills

Section two of this survey contained questions related to the technical skills that were developed during the project. All graduates responded to the same set of questions as they had all been involved in working on both the mobile application and the web application. The skills were categorised under the following areas:

- ASP.NET web development,
- SQL Server database management system,
- Application security,
- C#.NET and the .NET Compact framework,
- RDA data transfer.

Again, the graduates rated how much they learnt about that skill during the project and how useful the skill has been to them so far in their development career.

ASP.NET web development was one of the more useful technical skills for this cohort. Two graduates responded knowledge of this technology was useful and one responded that it had been essential (refer Figure 4.21). Two graduates answered they had learned a lot about this technology during the project, with one reporting they had learned the basics (refer Figure 4.21).

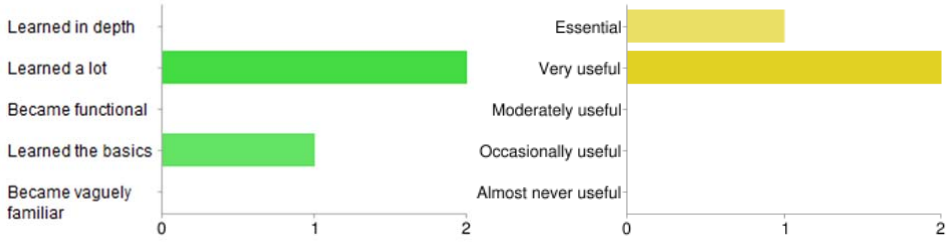


Figure 4.21: ASP.NET Web Development

The transferrable skills associated with use of the SQL Server database management system were seen as essential by one of the graduates, with the other two responding they had been very useful. Two graduates became functional with this technology and the other responded as having learnt a lot (refer Figure 4.22). Less development was done on the backend database during this iteration as the core structure and stored procedures that manipulated the data were already in place from the first iteration and were working well.

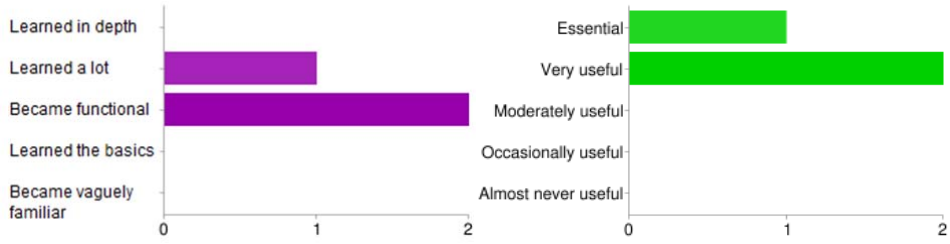


Figure 4.22: SQL Server Database Management System

Application Security covers things such as: user privileges, user authorisation, and data encryption. Two graduates believed they learned a lot about application security, with one reporting they had learned the basics. This skill has been essential for one graduate, and has proved useful to the other two respondents (refer Figure 4.23).

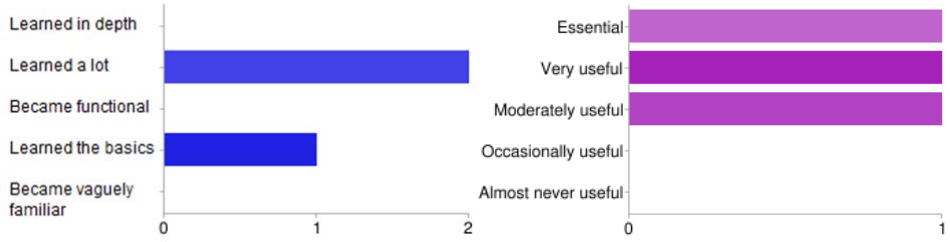


Figure 4.23: Application Security

As discussed in section 4.3.3.2, the C#.Net skills learnt by the graduates would be transferrable to other Microsoft .Net solutions; the .Net Compact framework would only be of use if developing windows mobile applications. The graduates felt they had learned a lot or became functional with these development skills, one had found them to be essential and the other two reported as very useful and moderately useful (refer Figure 4.24).

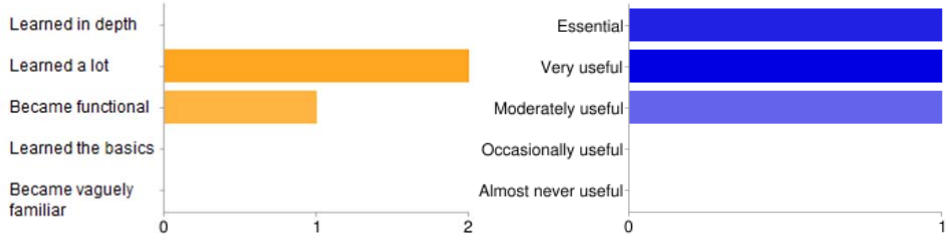


Figure 4.24: C#. Net and the .Net Compact Framework

The final technical skill evaluated was RDA data transfer (see section 4.3.3.2 for detail). All three respondents felt they learned a lot about this skill, with two reporting it to be very useful and one rating it only moderately useful (refer Figure 4.25).

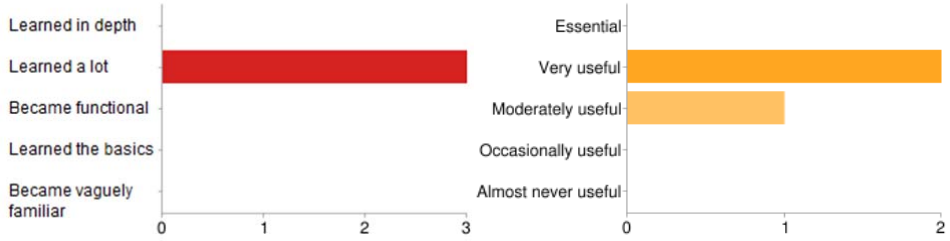


Figure 4.25: RDA data transfer

Further comment by one of the graduates affirmed the usefulness of working on a mobile application:

> *I found it useful to create an application for a mobile platform. There were functional differences and limitations imposed by the OS [operating system] and compact database not encountered in the desktop forms/browser applications I had been taught up to that point*

4.4.4 Real-Life Project Experience

The final section of questions related to the real-life project experience. Like the first iteration of the project all of the graduates felt their confidence as a developer had improved due to the experience (refer figure 4.25).

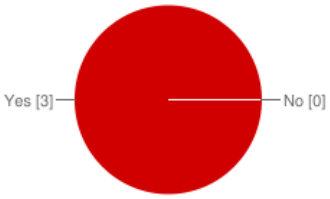


Figure 4.26: Improved confidence as a developer

One comment made was of particular interest to the researcher. A graduate who had been employed to work in a development team for a large organisation spoke of reflecting on the project two years into his role to help him maintain perspective when his confidence was flagging:

> *Six months after being accepted by [my employer] when I was feeling overwhelmed by the complexity, size, scope of software and processes at [my workplace], coupled with feeling inadequate when compared to the experience, skills, knowledge base of peers, I sat myself down and upon reflection discovered that I was in essence working with the MIT project core architecture. The capturing of user data,*

transporting the data through different software layers to remote storage, the use of web applications to manipulate the data. The MIT application also taught code reviews, peer reviews, and team collaboration. From my experience The MIT application most closely represents the real world, be it on a smaller less complex scale. I wish I had paid more attention to some of the processes. 18 months to 2 years later I still think back to the MIT application to help maintain perspective

Two of the graduates used the project as evidence of experience on their curriculum vitae (refer Figure 4.27). One of the graduates referred to the project experience in their job interview (refer Figure 4.28). This graduate used it as evidence of use of a particular technology.

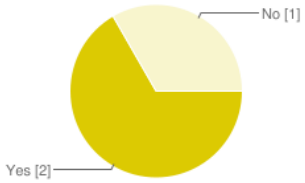


Figure 4.27: Project used as experience on CV

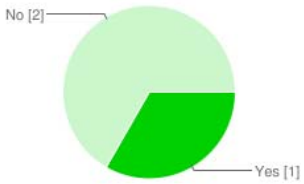


Figure 4.28: Project used as evidence of experience in interview

A further comment made by one graduate further validated the usefulness of the team experience:

I found that the MIT app gave a good insight into the way software is developed in a team environment, how all the parts can be worked on separately and bought together for testing or release

4.4.5 Skill Gaps

Only two skill gaps were identified for this iteration. One graduate expressed a need to utilise Microsoft Team Foundation Server, a more advanced form of source control that allows for reporting on project progress as well as the central code repository and version control. The other graduate commented they would like to learn more about estimation of resources required to complete a task.

4.4.6 The Redesigned Learning Environment

Estimation is difficult for even experienced development teams; it is a skill that develops over time. The agile method SCRUM (not used in iteration two of the development project)

makes estimation easier as the development cycles are short and estimation just has to be made for two-three weeks of development at a time. The researcher decided to re-introduce this method for the next iteration of the project, irrelevant of how many students were enrolled in the course (the techniques were dropped out due to the very small group in iteration two). The researcher implemented Team Foundation Server and added Team project support to Visual Studio 2008. This involved an update to Visual Studio 2008 to allow access to the Team Foundation Server tools through the development environment. Team Foundation Server had additional tools available to support the SCRUM development method so these were added to the basic configuration. The learning environment now had a tool which provided team web portals where work tasks could be itemised, assigned, and discussed as well as source control and conflict resolution. This allowed a more streamlined process for team communication, rather than having to use another environment to discuss work to be completed (i.e. the group discussion forum). As well as these additions and updates, the existing code was even more complex after the restructuring of the application. This would provide additional challenges around orientation within existing code.

4.5 Learning Environment Iteration Three

The re-designed learning environment was tested in the third iteration of the project (Semester one, 2011). Prior to semester start the desktop computers in the lab were migrated from Windows XP to Windows 7; also a newer version of the .Net framework was deployed. Neither of these upgrades had any impact on the application code. For this iteration there were 21 students enrolled, with 19 students volunteering to be involved in the research project. Due to the added complexity of the application and the larger cohort of students (five development teams) the researcher decided to issue a 'rescue' card to each group. The rescue card, labelled "Call in the chief architect", could be used once during the semester when the group felt as though they were stuck and needed assistance from the researcher. This was to aid the researcher in identifying the groups that were struggling, something that is easy to observe when working with ten or less students but not so easy when the class size is over 20.

4.5.1 Observations and Student Reflections

Issues and problems that were mentioned in the student's reflective journals, or observed by the researcher, during this iteration (refer Figure 4.29) were categorised using the following coding frame:

- Team-based development environment,
- SCRUM development method,
- Orientation of large existing code base,
- Team conflict,
- Time management.

The use of Team Foundation Server (TFS) caused the most problems for this cohort of students, with orientation within the existing application code the next most frequently mentioned issue (refer Figure 4.29). The complexity of TFS was mentioned throughout the semester in the reflective journals. Orientation within the existing code base was mentioned frequently in the initial weeks of the project (refer Figure 4.29). A comment in one student journal stating: “it’s like reading a book in another language”.

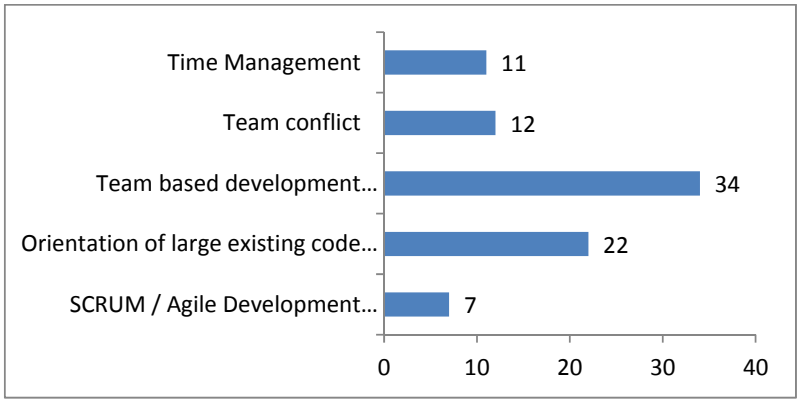


Figure 4.29: Issues identified in Reflective Journals / Class Observations

All teams appeared to have a good understanding of what was required in terms of the Agile method SCRUM. All teams were observed performing daily stand-ups, the initial meeting held at the start of each development session. Peer programming was embraced by a majority of the students with the more confident developers in the team ‘buddying up’ with developers who were less sure of their abilities. In terms of the client interaction techniques used in the Agile methods, there was only one team who regularly reported on progress and demonstrated features to the researcher, in the role of the product owner. The researcher observed two of the development teams getting stuck on one coding task for the last three weeks of semester. This has been recorded under time management issues as the teams that became stuck ran out of time to demonstrate complete features to the researcher.

In the final week of semester the researcher held a “project post-mortem”, a technique used by software development teams where everyone working on the project gets together and reflects on how things went and how it could be improved for next time. The two teams who appeared to be stuck were asked why they did not use the recuse card that was available to them. Both of the team’s responses were similar, they did not want to use up their one chance for help because then it would no longer be there. One team opted to not use any TFS features other than the source control aspect that was demonstrated by the researcher at the beginning of the development project. When questioned at the end of semester about this the group explained they felt the tool was too complicated and spending time making changes to the code was more important than using TFS. When asked if they had accessed any of the support materials provided to help them become familiar with TFS they advised they had not. Only one team made full use of the features available to them within TFS,

when asked about their experiences with using the tool it turned out that one group member had put in some effort outside of class to become familiar with the tool and had been almost solely responsible for the additional use.

4.5.2 Strategies for future iterations

Based on the observations and student reflections, the researcher devised strategies to help students in future iterations of the project. The first strategy involved changes to the development project assignment task. The mark allocation and wording of the assessment was altered to move the emphasis from code fixes to use of the team-based development tools and techniques. Additionally, the researcher created a screencast demonstrating the source control aspect of TFS. There were multiple comments in the reflective journals regarding the initial demonstration of TFS, many of the students felt overloaded with information in that session and struggled to follow along with the instructions. The ability to review the material that was presented in class after the initial introduction session will alleviate this issue. There was also a noted reluctance to use the external resources provided by the researcher to assist with learning this tool, the information is complex (as is the tool) and takes some time to work through. Finally, it was decided to issue teams with two rescue cards to address the reluctance to use their “one chance for help”.

4.6 Employer Survey Overview

A convenience sample of regional ICT employers was obtained from employers of the surveyed graduates and organisations known to the researcher through the Bachelor of Information and Commmunications Technology Industry Advisory Committee. The employer surveys were designed to determine the answers to the following research questions:

- Do non-technical skills (e.g. communication, problem solving and decision making, self-management, and teamwork) continue to be of high importance to employers hiring ICT graduates?
- Are the development languages, technologies, and processes utilised in the project still currently in demand in local industry?

Questions for the survey were created using categories adapted from previous local studies (Bekesi & Gardner, 2003; Snell et al., 2002; Young et al., 1999) and skill surveys conducted by international recruitment agencies and IT consultants (Argent, 2006; Watson, 2010). This provided a broad set of skills to which employers rated how important they thought each was for graduate IT workers in their organisation to possess. IT teams are often very diverse in terms of what workers might do as part of their daily routines, limiting the questions to only software development techniques and soft skills would make the questionnarie seem incomplete.

A total of eight responses was obtained from the ten employers invited to participate. Six of the employers had 20 or more IT employees, with the other two respondents coming from organisations with smaller IT teams. Of the smaller teams, one reported as having between one to five IT employees and the other as between five and ten. The respondents represented a cross section of both the public and private sector. Two employers are government ministries, one employer from the Education sector, one representative from a manufacturing industry, with the remaining four employers being IT companies (two software development and two network / system support (consultancy, design and support for small businesses))

4.6.1 Employer Survey Results

The employers were asked to rate a variety of different skills, languages, and technologies under the following categories:

- OO programming,
- 3GL procedural programming,
- User support & communication tools,
- Analysis & design,
- Database administration,
- Database querying and reporting,
- Team development tools,
- Project management,
- Operating systems,
- Internet /ecommerce,
- Networking & network administration,
- Mobile application development,
- Soft skills.

For each of the skills listed in the above categories, employers rated how important they thought this skill to be for graduate IT workers. The rating scale used was: Unimportant, Of Little Importance, Moderately important, Important, and Very important.

General desktop development was addressed first with employers being asked to rate the following OO programming skills:

- C#.Net,
- VB.Net
- Java.

The responses were widespread, six of the eight respondents rated both Java and C#.Net as being important (refer Figure 4.30: OO Programming - VB.Net, Java, C#.Net). VB.Net was

only slightly less important with five respondents selecting either moderately important or important (refer Figure 4.30).

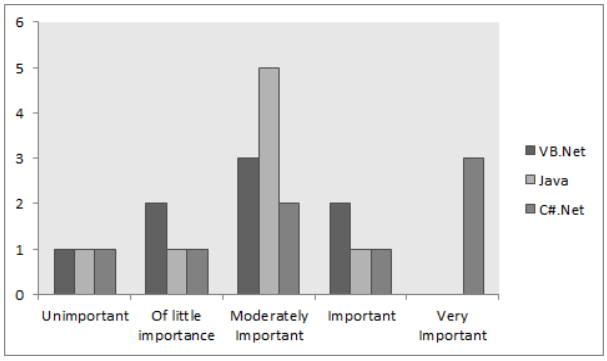


Figure 4.30: OO Programming - VB.Net, Java, C#.Net

The older procedural programming techniques associated with the third generation of computer languages, such as Delphi / Pascal and Visual Basic (VB), were not as highly rated as the newer Object-Oriented languages. Five respondents felt as though Delphi was unimportant, three rated it as moderately important. VB fared slightly better with half of the respondents rating it as important or moderately important (refer Figure 4.31).

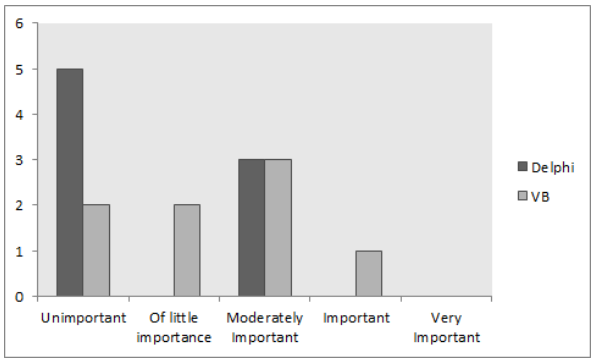


Figure 4.31: 3GL procedural programming skills – Delphi/Pascal, VB

The skills categorised under user support and communication tools (refer Figure 4.32) were:

- email/chat/office communicator,
- user training and support,
- use and evaluation of new software packages.

Seven of the respondents thought email/chat skills were important with only one employer feeling that these skills were of little importance. Three of the employers reported user training as a skill of little importance, the remaining five felt the ability to train was important or very important. A majority of the employers thought the ability to use and evaluate new software packages was important with only one responding this was of little importance (refer Figure 4.32).

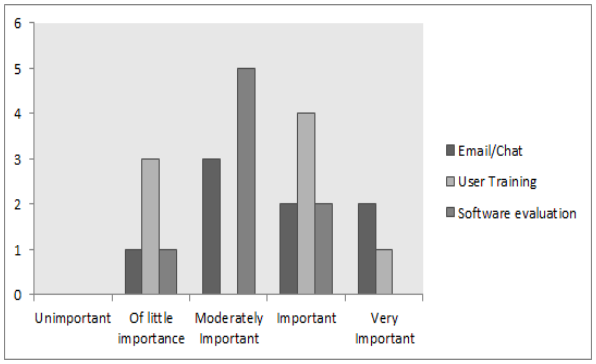


Figure 4.32: User support and communication tools

The Analysis and Design category (refer Figure 4.33) contained three different skill sets:

- Structured analysis & design,
- Object oriented (OO) analysis and design,
- Feasibility analysis

Structured analysis techniques are the traditional way of modelling information system requirements. The examples of structured models provided to the respondents were: dataflow diagrams and entity relationship diagrams; modelling the business processes and the structure of the data required by these processes, respectively. OO analysis is the more modern approach to analysis with a majority of the models in this toolset being created for the OO developers who will build the software solution. Tools from both structured analysis and OO analysis are often combined in projects to provide different views of the system to different stakeholders (e.g. customer, developers, and test analysts). Feasibility analysis occurs prior to the decision to build or buy a system and investigates whether the project is feasible from an organisational, economic, and technological standpoint.

Opinion was divided over the structured analysis methods with three employers rating them of little importance and the remaining five selecting important or very important. Half of the respondents rated OO analysis and Feasibility analysis as being moderately important, OO was considered of little importance by two employers. Only one respondent thought feasibility analysis was not an important skill set (refer Figure 4.33).

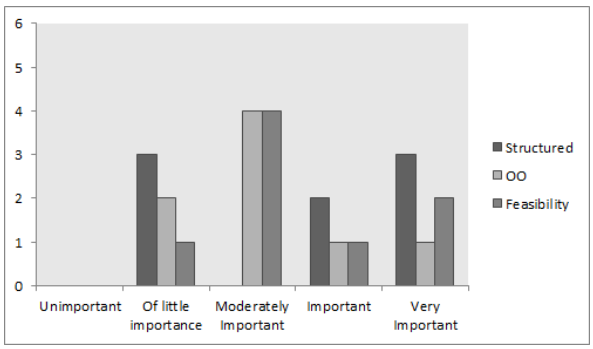


Figure 4.33: Analysis & Design - structured, OO, feasibility

Four of the most popular Relational Database Management Systems (RDMS) were listed under the database administration category. Two of the RDMS were enterprise level systems: Microsoft SQL Server and Oracle. The other two RDMS, Microsoft Access and MySQL, are able to be scaled to enterprise systems but generally are used by smaller businesses or as storage for data from a web application, respectively. MS SQL Server and MySQL were considered the most important, with Oracle seen as unimportant by a majority of the employers (refer Figure 4.34).

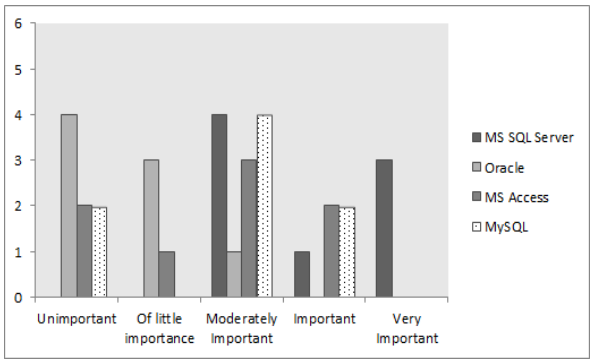


Figure 4.34: Database Administration - SQL Server, Oracle, Access, MySQL

Under the category of database querying and reporting (refer Figure 4.35), employers were asked to rate the skills of:

- Data warehousing,
- SQL Server Reporting Services,
- Language Integrated Querying (LINQ).

Data warehousing involves skills where information from disparate data sources is bought together into one database. SQL Server Reporting Services is an extra toolset for MS SQL Server that allows advanced reporting queries and display. LINQ allows for construction of database queries directly with the application code. All employers responded that SQL Server Reporting Services had a degree of importance whereas data warehousing was considered of little importance or unimportant by five of the eight respondents. LINQ, the newest of the data querying and reporting techniques, was reported as having some degree of importance by five of the eight respondents (refer Figure 4.35).

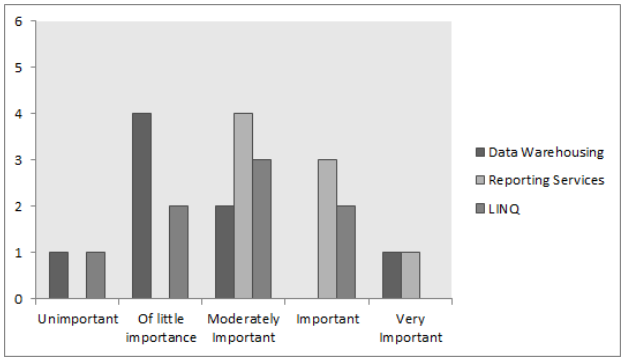


Figure 4.35: Data Querying and Reporting - Data warehousing, SQL Server Reporting Services, LINQ

Employers were asked to rate the following team development tools: source code repository, team portals or forums. Seven of the eight respondents felt these skills were of some importance (refer Figure 4.36).

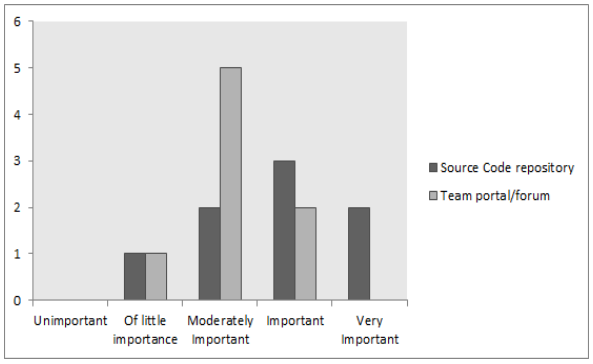


Figure 4.36: Team development tools - Source code repository, Team portal/Forum

The project management category contained skills based on:

- the Project Manager's Body of Knowledge (PMBOK),
- Agile methods,
- PRINCE 2.

Of these only the Agile methods were considered to be of some degree of importance by a majority (six) of the respondents (refer Figure 4.37).

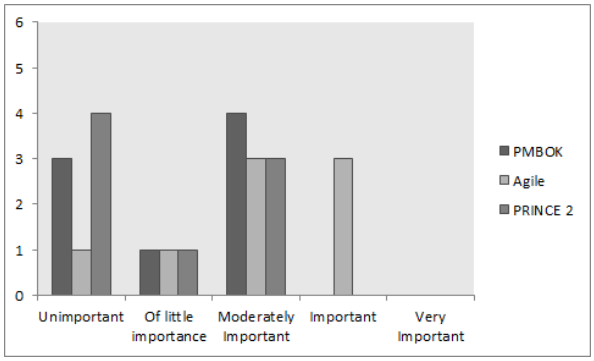


Figure 4.37: Project management techniques - PMBOK, Agile, Prince 2

In the operating systems administration category were the main Windows operating systems currently in use and a Linux based operating system (e.g. Ubuntu) (refer Figure 4.38). Of the Windows operating systems the older versions, XP and Server 2003, were considered the least important with two respondents rating them as unimportant or of little importance. Server 2008 and Windows 7 were rated of some importance by six employers, with Server 2008 being the only operating system to be given a very important rating. Knowledge of the Linux based operating system was considered of some importance by only three of the employers (refer Figure 4.38).

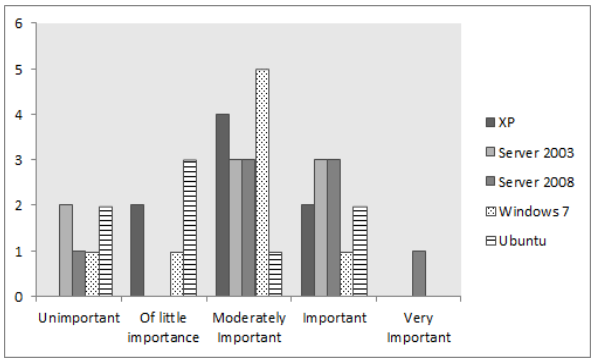


Figure 4.38: Operating System administration

The networking category covered a large skill set. For reporting purposes the researcher has divided these skills into two areas: network & network administration. In the network category (refer Figure 4.39), respondents were asked to rate:

- Cisco Certified Network Associate (CCNA),
- Internet Protocol V6 (IPV6),
- Internet setup and security,
- Network security,
- Voice over IP (VOIP),
- Wireless networks.

CCNA is considered an entry-level networking certification, only two of the employers thought this was important. IPV6 is the new Internet addressing scheme said to be of paramount importance to implement to ensure the Internet does not run out of IP addresses, three of the employers thought IPV6 skills were of some importance. All of the respondents felt Internet setup and security was of some importance, with five rating it as important and three as moderately important. Six employers felt network security was of some importance, with one employer rating it as very important. VOIP and wireless networks were both considered of some importance by half of the respondents (refer Figure 4.39).

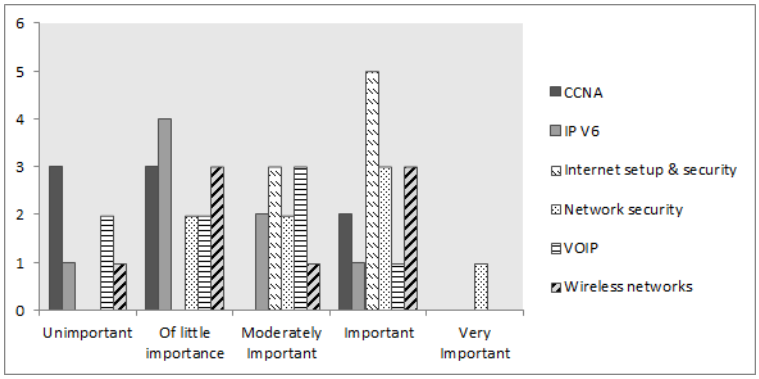


Figure 4.39: General networking skills – CCNA, IP V6, Internet setup & security, Network security, VOIP, Wireless

The network administration category contained skills associated with server and desktop management:

- Virtual Servers,
- Virtual Desktop Infrastructure (VDI),
- Thin Client,
- Storage area networks (SAN).

Virtual Servers and Thin Client were considered the most necessary skills, with five of the employers rating them of some importance (refer Figure 4.40).

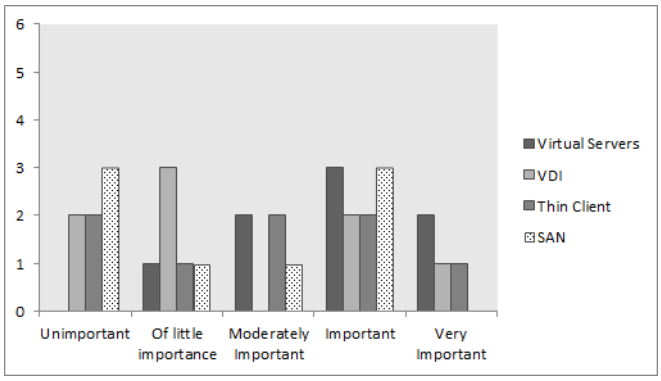


Figure 4.40: Network Administration - Virtual Servers, VDI, Thin Client, SAN

The skills assessed in the Internet and Ecommerce category (refer Figure 4.41) were:

- ASP.Net,
- PHP,
- Java Web,
- Web Services,
- Cascading Style Sheets (CSS)
- JavaScript.

All of these are different languages used to develop web applications, other than web services which is a method of communication between devices over the internet. ASP.Net, Web Services and CSS / JavaScript were considered to be of some importance by a majority of the employers, with only one employer rating CSS &JavaScript as unimportant. PHP and Java Web skills were considered unimportant or of little importance by half of the respondents (refer Figure 4.41).

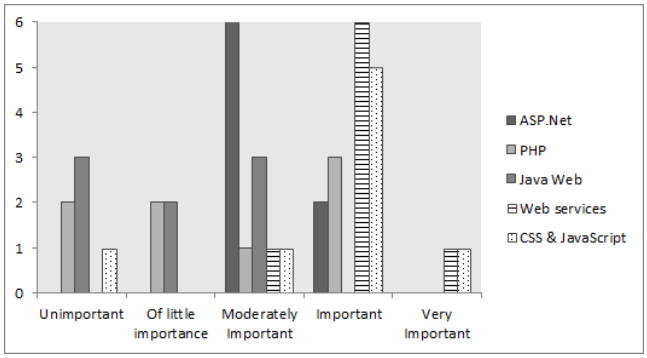


Figure 4.41: Internet/Ecommerce - ASP.Net, PHP, Java web, Web services, CSS & JavaScript

Employers were asked to rate the mobile application development skills associated with developing for:

- .NET compact framework,
- Java 2 Mobile Edition (J2ME)
- Android devices.

Half of the respondents felt the .NET compact framework was moderately important. J2ME was rated as moderately important by three of the employers, with Android receiving two votes for moderately important (refer Figure 4.42).

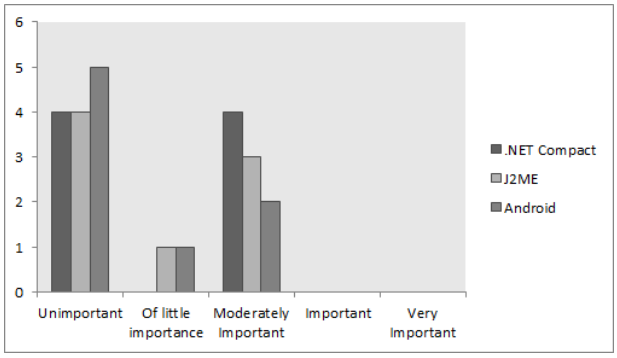


Figure 4.42: Mobile applications - .Net Compact, J2ME, Android

The soft skills category (refer Figure 4.43) contained a wide range of interpersonal and intrapersonal abilities:

- communication,

- creative thinking,
- global mind-set,
- problem solving,
- initiative and enterprise,
- information gathering,
- self-management,
- perseverance,
- team working skills.

Opinion was less divided in the soft skills category. The employers represented a diverse cross-section of business so the differing opinion on the importance of the technical skills was expected. The ratings associated with the soft skills were less likely to be effected by the business domain the organisations are working within. This was reflected in the responses with seven of the eight employers rating all soft skills bar one, global mind-set, as either important or very important. Global mind-set was the only soft skill to receive an unimportant rating from one employer, three employers considered it moderately important with the remaining four rating it as important (refer Figure 4.43). Honest self-assessment and documentation were two other soft skills suggested as necessary.

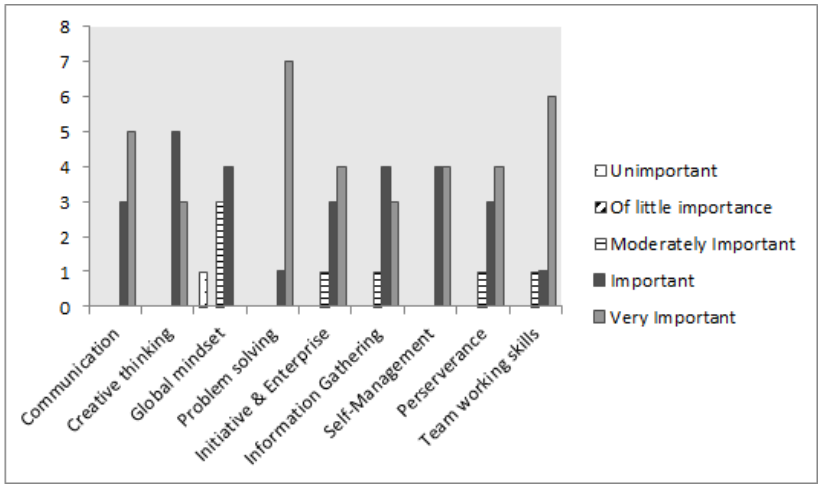


Figure 4.43: Soft Skills

4.7 Summary

Graduates from two iterations of the real-life, software development project were surveyed to find out which skills they felt were developed during the project and how useful these skills have been to them so far in their careers. For graduates of the first iteration: peer-programming, the .NET framework, data encryption techniques, and database querying have been the most useful skills developed during the project. Graduates from iteration one identified several gaps in their skill sets, most of which had already been addressed in other BICT courses. The skill that had not been addressed, using source control software, was

introduced into the learning environment for the next iteration of the project. Other changes to the learning environment for iteration two were: that there was now a large existing codebase the students would need to orient themselves within before any modifications could be made to the code, there was feedback available on bugs and annoyances found by users during the first pilot test, and some technology updates had occurred.

In iteration two the student cohort was smaller and peer-programming was not viable. Team communication was supported by a discussion forum, rather than use of Agile project management methods and peer-programming techniques. For the iteration two cohort the team-based tools, and experience with team dynamics were considered the most useful skills learnt by all of the respondents. Feedback on skill gaps from the second graduate survey led to the researcher introducing the more advanced team development tool of Team Foundation Server to the learning environment and ensuring that further iterations of the project utilised the Agile project management methods and peer-programming techniques that were present in iteration one.

Both sets of graduates found the real-life aspect of the project experience to be invaluable during their job interviews and most used it as evidence of experience on their CVs. One graduate even commented that they still reflected on the experience eighteen months to two years into their employment when they doubted their ability to handle the complexity within their work environment.

In the third iteration of the project, the researcher's observations were used alongside the student's reflective journals to identify difficulties and issues experienced when using the redesigned learning environment. The use of Team Foundation Server caused the most problems for this cohort of students, with orientation within the existing code the next most frequently mentioned issue. The researcher devised strategies to help overcome the difficulties including: reallocation of marks within the assessment task to place more emphasis on the team development tool use, and creation of additional resources to aid with learning the source control aspect of the tool.

A selection of regional employers were surveyed to determine whether the development languages and technologies utilised in the project were still in demand, and to see if non-technical skills were still of high importance when hiring ICT graduates. The responses to the technical skill sets were varied, and quite dependent on the particular IT specialisation required by each organisation. However, the core techniques used within the project were considered important by a majority of the respondents. These were: C#.Net, ASP.Net, Agile project management, and the database skills associated with SQL Server. There was no such spread of results when employers were asked about non-technical (soft) skills. With the exception of global mind-set, all soft skills were considered to be important or very important by all employers.

CHAPTER FIVE

5.1 Introduction

In this chapter, an overview of the thesis is presented, as well as the major findings in answer to the research questions. The significance of the study is then described, followed by the limitations. The chapter concludes with recommendations for further research and final comments.

5.2 Overview of thesis

This study aimed to validate the relevance of skills acquired during a real-life, team-based development project at preparing graduates for junior developer positions. Graduates of two iterations of the researcher's third year Software Engineering course who were employed in junior developer roles were surveyed in order to determine the depth of skills developed within this learning environment and to try and identify common gaps in their skill sets experienced as new employees. The feedback from the graduates was used to improve the learning environment provided to the software engineering students on the Bachelor of Information and Communications Technology (Applied) degree with the aim of making graduate transition to the workplace more seamless. The employers of the graduates were surveyed to identify the core skills required of their new ICT graduates to confirm the relevance of an adapted learning environment. The final iteration of the research study involved classroom observations of students working within the adapted environment. The observations were used alongside the students' reflective journals to identify the challenges experienced working within the adapted environment to allow the researcher to aid students to overcome these hurdles in future iterations of the real-life, team-based project.

Chapter 1 introduced the premise on which the study was based, that software engineering education has been lacking at teaching the skills that will make graduates work ready, with junior developers being ill-prepared for the type of work they will likely encounter, that of software maintenance and bug-fixing. The context of the study was described and the research questions were outlined, this was followed by a description of the research methods and the significance and limitations of the study.

Chapter 2 reviewed the literature surrounding software engineering education, beginning with the debate that software development is an engineering practice not a science discipline. The traditional approach to education and the demands of the knowledge society were described, followed by an overview of the history of software developer education. Industry perspectives on the skills required by junior developers were detailed and modern techniques used within courses to try and teach industry practices were examined. The chapter concluded by suggesting that a real-life, team-based project approach would bridge the gap between academic theory and real world practice.

Chapter 3 presented an overview of the research study. The title of the research study was stated and the significance of the study described. Then the research questions were restated and the research design and methodology were outlined, including the sampling and distribution. The learning environment within which the study took place was also described, as were the approaches to collecting and analysing the data

Chapter 4 presented the qualitative analyses. Results from the two samples of graduate surveys were presented, followed by a categorisation of common issues appearing in the student reflective journals that were kept during the third iteration of the project where students were utilising the adapted learning environment. Based on this, the researcher then made recommendations for further iterations of the project that should overcome some of the issues experienced. Finally, the employer responses on graduate skill requirements were presented.

5.3 Major findings

The findings are summarised in order of the research questions as presented in Chapters 1 and 3 of this study.

Research Question One: Do non-technical skills (e.g. communication, problem solving and decision making, self-management, and teamwork) continue to be of high importance to employers hiring ICT graduates?

The soft skills category in the employer survey covered a wide range of interpersonal and intrapersonal abilities: communication, creative thinking, global mind-set, problem solving, initiative and enterprise, information gathering, self-management, perseverance, and team working skills. The employers were asked to rate skills using the following scale: unimportant, of little importance, moderately important, important, and very important. Though the respondents represented a diverse cross-section of industry there was very little diversity in the results. Seven of the eight employers rated all soft skills bar one, global mind-set, as either important or very important. Global mind-set received an unimportant rating from one employer, three employers considered it moderately important with the remaining four rating it as important. The results reflected that soft skills are important to all employers of ICT graduates. Honest self-assessment and documentation were two other soft skills suggested as necessary.

Research Question Two: Are the development languages, technologies, and processes utilised in the project still currently in demand in local industry?

The technical skills demonstrated in the project were: ASP.Net, Microsoft SQL Server database management and querying, C#.Net, Agile project management, working within a Windows environment (client and server), source code control, team communication forums, and Windows Mobile application development using .Net Compact Framework and the

Compact Edition database. Due to the respondents coming from a diverse cross-section of industry it was expected that results on the importance of specific technical skills would differ based on the core business of the employer's organisation.

Of the eight employer respondents, six considered ASP.Net to be moderately important, with the remaining two respondents rating it as important.

Similarly, Microsoft SQL Server was seen as important by all respondents with four ratings of moderately important, one rating of important, and three very important ratings.

Six employers rated C#.Net as having some level of importance. Two employers rated it of moderate importance, one as important, and three rated it as very important.

Agile project management was the only project management technique listed in the survey that was considered to be of some degree of importance by a majority of the respondents (six).

Skills developed working within a Windows environment in particular the newer versions of Windows 7 client and Windows 20008 server were rated of some importance by six respondents. Windows 2008 was the only operating system to be given a very important rating.

The team support tools of source code repository (source control) and team forums were considered of some importance by seven of the eight respondents. Five respondents rated source code repository as being important or very important.

The mobile application development skills came across as the least important, with the .Net Compact framework rated as moderately important by half of the respondents and unimportant by the remaining respondents. At the time of data collection mobile applications were cutting-edge technology with only a small number of organisations developing for, or supporting mobile platforms. The researcher believes if the employer survey was administered again, given the saturation of mobile apps and mobile devices now on the market, the result would be different.

Research Question Three: What skills did the graduates perceive as improved or formed by participating in the real-life development project?

Other than some prior knowledge of the Transact_SQL database query language (gained in a previous course) and personal use of online forums, all of the technical skills and team-based development skills utilised within the project were new to both iterations of the graduates.

In iteration one, all of the graduates responded as becoming functional or learning a lot about: Agile project management with SCRUM, peer programming, ASP.Net web development, SQL Server database, RDA data transfer with the Compact edition database,

and mobile application security. One of the mobile developers felt as though they had only learned the basics of C#.Net and the .Net Compact framework. This was the developer who took the support role of navigator in a majority of the peer programming sessions. The researcher ensured that roles were swapped regularly during the third iteration when students were again using peer programming methods. The skill that the web developers learnt the least about was web application security, with two reporting they learned the basics and one respondent saying they had become functional. There was less emphasis on the security of the web application as it was going to be hosted on the UCOL intranet (a site only available internally) and used to monitor and report on student progress only.

In iteration two, all of the graduates responded as becoming functional or learning a lot about: team dynamics, working with a large existing codebase, SQL Server database, C#.Net and the .Net Compact framework, and RDA data transfer. Responses on the use of the code repository software supporting version control were mixed; one graduate became vaguely familiar, one learned the basics, and the third reported learning a lot. Discussion forum use for team communication received a rating of learned a lot from two of the graduates, the third responded as learning the basics. The two graduates who reported they had learned a lot were responsible for driving the restructuring of the code into a multi-tier application and used the forum heavily to discuss this. Two of the graduates felt they had learned a lot about ASP.net web development and application security, the third felt they had learned the basics.

Research Question Four: Of the skills demonstrated during the project which were the ones required of the graduate developers in their first ICT role?

For graduates of the first iteration of the project: peer-programming, interaction with a real client, the .NET Compact framework, data encryption techniques, and database querying were the most useful skills developed during the project. Agile project management has also been very useful to three of the five respondents but one respondent has never found these skills to be of use.

For the iteration two cohort: the experience with team dynamics, orienting themselves within a large existing codebase, ASP.Net web development, and SQL Server were considered the most useful skills learnt by all of the respondents. All of these skills were seen as essential or very useful. The team-based tools (code repository, team forum) were seen as essential or very useful by two of the respondents, but had not been much use to the third respondent. The third respondent was working as an application support specialist for a non-IT business so was working more in isolation than were the other two respondents.

Research Question Five: Did the learning environment provide real-world experience that the novice developers could use as evidence of skills during employment interviews?

Both sets of graduates found the real-life aspect of the project experience to be invaluable during their job interviews and most used it as evidence of experience on their CVs. One graduate even commented that they still reflected on the experience eighteen months to two years into their employment when they doubted their ability to handle the complexity within their work environment. A majority of the graduates who used the project experience as evidence in job interviews provided it as an example of experience using a particular technology. All graduates responded that their confidence as a developer was improved by the experience.

Research Question Six: Are there common themes amongst the skills the graduates identify as lacking in their knowledge base that could be addressed by adaptation of the learning environment?

Graduates from iteration one identified several gaps in their skill sets, most of which had already been addressed in other BICT web and database related courses (JavaScript, JQuery, LINQ, XSLT and XML). The skill that had not been addressed, using source control software, was introduced into the learning environment for the next iteration of the project. Other changes to the learning environment for iteration two that made the project even more realistic were: there was now a large existing codebase that the students would need to orient themselves within before any modifications could be made to the code, there was feedback available on bugs and annoyances found by users during the first pilot test.

Only two skill gaps were identified by the iteration two cohort. One graduate expressed a need to utilise Microsoft Team Foundation Server, a more advanced form of source control that allows for reporting on project progress as well as the central code repository and version control. The other graduate commented they would like to learn more about estimation of resources required to complete a task. Team Foundation Server was implemented within the learning environment for iteration three, and the Agile project management method SCRUM was re-introduced. Agile methods make estimation for novices easier as the development cycles are short and estimation just has to be made for two-three weeks of development at a time.

5.4 Significance

This research validated and improved the learning environment provided to the software engineering students on the Bachelor of Information and Communications Technology (Applied) degree and has made graduate transition to the workplace more seamless.

It found that the skills being taught within the learning environment are useful for a majority of the graduates in their junior developer roles. The real-life project has provided graduates with experience in team-based development and particular technologies that they have used as evidence on their CVs and during their job interviews. In particular, the soft skills developed during the project meet the requirements of the regional ICT community.

The information on the learning environment and techniques used to help students overcome difficulties faced in the classroom have added to the body of knowledge on tertiary teaching methods, particularly in relation to technology-based courses.

The research on the effectiveness of the learning environment will be of use to other institutions in New Zealand looking to implement a similar approach.

5.5 Limitations

Limitations of this study are to do with the small sample size. Small class sizes for the first two iterations of the learning environment led to a small sample of graduates (eight responses from a possible eleven graduates) and therefore this led to a small employer sample (eight responses). Nevertheless, indicating results are still valuable from this small sample.

5.6 Suggestions for further research

A similar study could be carried out with graduates of the learning environment not included in the graduate survey sample to improve on the small sample size. For example, the class size for the third iteration involved 19 participants. After identification of other Polytechnics within New Zealand offering a similar learning environment, a multi-institute study could be carried out to increase the sample size. This would also provide employer data that was less regionally specific. Due to the rapid changes that happen with technology, employers could be surveyed on an annual basis to ensure continued relevancy of skills being developed through the real-life, team-based project. It would also be beneficial to survey the employers to determine if the graduates from the learning environment were more work ready than other graduates hired by the organisation who did not have a real world, team-based development experience during their studies. Identification of high performing teams within the learning environment and analysis of communication strategies and techniques adopted by those teams would be an area of study that could aid future participants of the learning environment.

5.7 Final comments

This study validated the concept that real-life, team-based software development projects do indeed help to bridge the gap between the academic world and what is required of junior developers in industry. The skills developed in the project, and in particular the soft skills are

in demand by regional IT employers and graduates of the learning environment have used the experience as evidence during the job seeking process.

Based on the positive outcome for the graduates of this study, the researcher is investigating the restructuring of the software development courses within the BICT degree to provide a more sustained real-life project experience. Ideally, students will begin their real-life project experience in their second semester of second year coming in as junior developers into an existing project team. The existing project team will contain third year students working as intermediate and senior developers who are in their second and third semester of the real-life project experience, respectively. Junior developers will be mentored by senior developers who will aid them with their orientation into the learning environment. The more sustained nature of this approach will provide ample time for students to become familiar with the tools, technologies and processes utilised within the learning environment and should allow the students to contribute significantly to the software solution being developed.

References

Abran, A., Moore, J. W., Bourque, P., & Dupuis, R. (Eds.). (2004). *Guide to the Software Engineering Body of Knowledge* (2004 ed.): IEEE Computer Society.

Anderson, R. (2008). Implications of the Information and Knowledge Society for Education. In J. Voogt & G. Knezek (Eds.), *International Handbook of Information Technology in Primary and Secondary Education.* (Vol. 20, pp. 5-22): Springer US. Retrieved from http://dx.doi.org/10.1007/978-0-387-73315-9_1 doi:10.1007/978-0-387-73315-9_1

Ardis, M., & Ford, G. (1989). *SEI Report on Graduate Software Engineering Education (1989)* (CMU/SEI-89-TR-021). Retrieved from http://www.sei.cmu.edu/library/abstracts/reports/89tr021.cfm

Argent, G. (2006). Vital skills for today's professionals. *Computer Weekly*, 3-3. Retrieved from http://search.ebscohost.com/login.aspx?direct=true&db=buh&AN=21406398&site=ehost-live

Armour, P. G. (2004). Beware of counting LOC. *Commun. ACM, 47*(3), 21-24. doi:10.1145/971617.971635

Bauer, F. (1973). Software and Software Engineering. *SIAM Review, 15*(2), 469-480. Retrieved from http://epubs.siam.org/doi/abs/10.1137/1015067. doi:doi:10.1137/1015067

Begel, A., & Simon, B. (2008a). *Novice software developers, all over again*. Paper presented at the Proceeding of the Fourth international Workshop on Computing Education Research, Sydney, Australia.

Begel, A., & Simon, B. (2008b). *Struggles of new college graduates in their first software development job*. Paper presented at the Proceedings of the 39th SIGCSE technical symposium on Computer science education, Portland, OR, USA.

Bekesi, E., & Gardner, N. (2003). *Technical Skills needed in Software Development - A snapshot from 2002.* Paper presented at the 16th Annual Conference of the National Advisory Committee on Computing Qualifications, Palmerston North.

Blake, M. B. (2003). A student-enacted simulation approach to software engineering education. *Education, IEEE Transactions on, 46*(1), 124-132. Retrieved from 10.1109/TE.2002.808255

Boehm, B. W. (1976). Software Engineering. *Computers, IEEE Transactions on, C-25*(12), 1226-1241. doi:10.1109/tc.1976.1674590

Bogdan, R., & Biklen, S. (2003). *Qualitative research in education: an introduction to theory and methods* (4th Edition ed.). USA: Pearson Education Group Inc.

Borenstein, N. S. (1992). Colleges Need to Fix the Bugs in Computer-Science Courses. *The Chronicle of Higher Education, 38*(45), 2-B3. Retrieved from http://search.proquest.com/docview/214642018?accountid=10382

Brechner, E. (2003). *Things they would not teach me of in college: what Microsoft developers learn later*. Paper presented at the Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, Anaheim, CA, USA.

Brooks, F. P. (1996). The computer scientist as toolsmith II. *Commun. ACM, 39*(3), 61-68. doi:10.1145/227234.227243

Chase, J., Oakes, E., & Ramsey, S. (2007). Using live projects without pain: the development of the small project support center at Radford University. *ACM SIGCSE Bulletin, 39*(1), 469-473.

Cleland, S. (2003). *Agility in the classroom: Using Agile Development Methods to foster team work and adaptability amongst undergraduate programmers.* Paper presented at the 16th Annual NACCQ Palmerston North.

Cohen, L., Manion, L., & Morrison, K. (2000). *Research methods in education* (5th ed.). London: RoutledgeFalmer.

Coppit, D. (2006). Implementing Large Projects in Software Engineering Courses. *Computer Science Education, 16*(1), 53-73.

Dawson, R. J. (2000). *Twenty dirty tricks to train software engineers*. Paper presented at the Proceedings of the 22nd international conference on Software engineering, Limerick, Ireland.

Dawson, R. J., Newsham, R. W., & Fernley, B. W. (1997). Bringing the `real world' of software engineering to university undergraduate courses. *Software Engineering. IEE Proceedings- [see also Software, IEE Proceedings], 144*(5), 287-290.
Denning, P. J. (1992). Educating a new engineer. *Commun. ACM, 35*(12), 82-97. doi:10.1145/138859.138870
Dijkstra, E. W. (1968). Letters to the editor: go to statement considered harmful. *Commun. ACM, 11*(3), 147-148. doi:10.1145/362929.362947
Dijkstra, E. W. (1972). The humble programmer. *Commun. ACM, 15*(10), 859-866. doi:10.1145/355604.361591
Drucker, P. F. (1988). The coming of the new organization. *Harvard Business Review*(January-February 1988), 3-11.
Duggins, S. L., & Thomas, B. B. (2002, 2002). *An historical investigation of graduate software engineering curriculum.* Paper presented at the Proceedings of the 15th Conference on Software Engineering Education and Training, 2002. (CSEE&T 2002). .
Dupuis, R., Bourque, P., & Abran, A. (2003, 5-8 Nov. 2003). *SWEBOK guide an overview of trial usages in the field of education.* Paper presented at the Frontiers in Education, 2003. FIE 2003 33rd Annual.
Fauldi, A. (1973). *Planning theory*. Oxford: Pergamon Press.
Ford, G. (1994). *Progress Report on Undergraduate Software Engineering Education, A* (CMU/SEI-94-TR-011). Retrieved from http://www.sei.cmu.edu/library/abstracts/reports/94tr011.cfm
Frailey, D. J., & Mason, J. (2002, 2002). *Using SWEBOK for education programs in industry and academia.* Paper presented at the Software Engineering Education and Training, 2002. (CSEE&T 2002). Proceedings. 15th Conference on.
Freeman, P. (1987). Essential Elements of Software Engineering Education Revisited. *Software Engineering, IEEE Transactions on, SE-13*(11), 1143-1148. doi:10.1109/tse.1987.232862
Freeman, P., Wasserman, A. I., & Fairley, R. E. (1976). *Essential elements of software engineering education*. Paper presented at the Proceedings of the 2nd international conference on Software engineering, San Francisco, California, USA.
Friend, J., & Hickling, A. (1987). *Planning under pressure: The strategic choice approach*. Oxford: Pergamon Press.
Highsmith, J. (2002). Does Agility Work? *Software Development Magazine, 10*(6), 30.
Highsmith, J., & Fowler, M. (2001). The agile manifesto. *Software Development Magazine, 9*(8), 29-30.
Hilburn, T. B. (1997). Software engineering education: a modest proposal. *Software, IEEE, 14*(6), 44-48. doi:10.1109/52.636650
Hislop, G. W., Lutz, M. J., Naveda, J. F., McCracken, M., Mead, N., & Williams, L. (2002). Integrating Agile Practices into Software Engineering Courses *Computer Science Education, 12*(3), 169-185.
Hogan, J. M., & Thomas, R. (2005). *Developing the software engineering team*. Paper presented at the Proceedings of the 7th Australasian conference on Computing education - Volume 42, Newcastle, New South Wales, Australia.
Horning, J. J., & Wortman, D. B. (1977). Software Hut: A Computer Program Engineering Project in the Form of a Game. *Software Engineering, IEEE Transactions on, SE-3*(4), 325-330. doi:10.1109/tse.1977.231151
Huffman Hayes, J. (2002, 2002). *Energizing software engineering education through real-world projects as experimental studies.* Paper presented at the Software Engineering Education and Training, 2002. (CSEE&T 2002). Proceedings. 15th Conference on.
Humphrey, W. S. (1996). Using a defined and measured Personal Software Process. *Software, IEEE, 13*(3), 77-88. doi:10.1109/52.493023
Joy, M. (2005). Group projects and the computer science curriculum. *Innovations in Education and Teaching International, 42*(1), 15-25.
Kajko-Mattsson, M., Forssander, S., Andersson, G., & Olsson, U. (2002). Developing CM3: Maintainers' Education and Training at ABB. *Computer Science Education, 12*(1-2), 57-89. Retrieved from http://www.tandfonline.com/doi/abs/10.1076/csed.12.1.57.8212. doi:10.1076/csed.12.1.57.8212

Kemmis, S. (2009). Action research as a practice-based practice. *Educational Action Research, 17*(3), 463-474. Retrieved from http://dx.doi.org/10.1080/09650790903093284. doi:10.1080/09650790903093284
Kessler, R., & Dykman, N. (2007). Integrating traditional and agile processes in the classroom. *SIGCSE Bull., 39*(1), 312-316. doi:10.1145/1227504.1227420
Koppi, T., & Naghdy, F. (2009). *Managing educational change in the ICT discipline at the tertiary education level: Final Report*. Retrieved from http://www.olt.gov.au/resource-managing-educational-change-ict-discipline-uow-2009
Larman, C., & Basili, V. R. (2003). Iterative and incremental development: A brief history. *Computer, June,* 47-56.
Lethbridge, T. C. (1998a). The relevance of software education: A survey and some recommendations. *Annals of Software Engineering, 6*(Dec), 91-110.
Lethbridge, T. C. (1998b, 22-25 Feb 1998). *A survey of the relevance of computer science and software engineering education.* Paper presented at the Software Engineering Education, 1998. Proceedings., 11th Conference on.
Lethbridge, T. C. (2000). What knowledge is important to a software professional? *Computer, 33*(5), 44-50. doi:10.1109/2.841783
Lethbridge, T. C., LeBlanc, R. J., Sobel, A. E. K., Hilburn, T. B., & Diaz-Herrera, J. L. (2006). SE2004: Recommendations for Undergraduate Software Engineering Curricula. *Software, IEEE, 23*(6), 19-25. doi:10.1109/ms.2006.171
Leventhal, L. M., & Mynatt, B. T. (1987). Components of Typical Undergraduate Software Engineering Courses: Results from a Survey. *Software Engineering, IEEE Transactions on, SE-13*(11), 1193-1198. doi:10.1109/tse.1987.232869
Lewin, K. (1946). Action Research and Minority Problems. *Journal of Social Issues, 2*(4), 34-46. Retrieved from http://dx.doi.org/10.1111/j.1540-4560.1946.tb02295.x. doi:10.1111/j.1540-4560.1946.tb02295.x
Ludi, S., & Collofello, J. (2001, 2001). *An analysis of the gap between the knowledge and skills learned in academic software engineering course projects and those required in real: projects.* Paper presented at the Frontiers in Education Conference, 2001. 31st Annual.
Maletic, J. I., Howald, A., & Marcus, A. (2001, 2001). *Incorporating PSP into a traditional software engineering course: an experience report.* Paper presented at the Software Engineering Education and Training, 2001. Proceedings. 14th Conference on.
Mar, K., & Schwaber, K. (2002). Scrum with xp. *InformIT,* (22 March). Retrieved from www.informit.com
Maurer, F., & Martel, S. (2002). On the productivity of agile software practices: An industrial case study. *Retrieved September, 20*, 2004. Retrieved from www.itu.dk/~katten/speciale/On%20the%20Productivity%20of%20Agile%20Software%20Practices-%20An%20Industrial%20Case%20Study.pdf
McConnell, S. (1998). The art, science, and engineering of software development. *Software, IEEE, 15*(1), 120, 118-119. doi:10.1109/52.646892
McDowell, C., Werner, L., Bullock, H. E., & Fernald, J. (2006). Pair programming improves student retention, confidence, and program quality. *Commun. ACM, 49*(8), 90-95. doi:http://doi.acm.org/10.1145/1145287.1145293
McMillan, W. W., & Rajaprabhakaran, S. (1999, 22-24 Mar 1999). *What leading practitioners say should be emphasized in students' software engineering projects.* Paper presented at the Software Engineering Education and Training, 1999. Proceedings. 12th Conference on.
Meziane, F., & Vadera, S. (2004, 1-3 March 2004). *A comparison of computer science and software engineering programmes in English universities.* Paper presented at the Software Engineering Education and Training, 2004. Proceedings. 17th Conference on.
Moore, M., & Potts, C. (1994). Learning by doing: Goals and experiences of two software engineering project courses. In J. Díaz-Herrera (Ed.), *Software Engineering Education.* (Vol. 750, pp. 151-164): Springer Berlin Heidelberg. Retrieved from http://dx.doi.org/10.1007/BFb0017611 doi:10.1007/BFb0017611
Moser, R., Abrahamsson, P., Pedrycz, W., Sillitti, A., & Succi, G. (2008). A case study on the impact of refactoring on quality and productivity in an agile team *Balancing Agility and Formalism in Software Engineering.* (pp. 252-266): Springer.

Newell, A., & Simon, H. A. (1976). Computer science as empirical inquiry: symbols and search. *Commun. ACM, 19*(3), 113-126. doi:10.1145/360018.360022
NZQA. (2012). *New Zealand Qualifications Framework Search Results - Information Technology, Bachelor* Retrieved from http://www.nzqa.govt.nz/studying-in-new-zealand/nzqf/
Parnas, D. L. (1999). Software engineering programs are not computer science programs. *Software, IEEE, 16*(6), 19-30. doi:10.1109/52.805469
Poger, S., & Bailie, F. (2006). Student perspectives on a real world project. *J. Comput. Small Coll., 21*(6), 69-75.
Postema, M., Dick, M., Miller, J., & Cuce, S. (2000). Tool Support for Teaching the Personal Software Process. *Computer Science Education, 10*(2), 179-193. Retrieved from http://www.tandfonline.com/doi/abs/10.1076/0899-3408%28200008%2910%3A2%3B1-C%3BFT179. doi:10.1076/0899-3408(200008)10:2;1-c;ft179
Postema, M., Miller, J., & Dick, M. (2001, 2001). *Including practical software evolution in software engineering education.* Paper presented at the Software Engineering Education and Training, 2001. Proceedings. 14th Conference on.
Reeve, D., & Petch, J. (1999). *GIS, organisations and people: A socio-technical approach*. London: Taylor & Francis.
Reichlmayr, T. (2003, 5-8 Nov. 2003). *The agile approach in an undergraduate software engineering course project.* Paper presented at the Frontiers in Education, 2003. FIE 2003 33rd Annual.
Schlimmer, J. C., Fletcher, J. B., & Hermens, L. A. (1994). Team-oriented software practicum. *Education, IEEE Transactions on, 37*(2), 212-220. doi:10.1109/13.284997
Schneider, J. G., Johnston, L., & Joyce, P. (2005, 29 March-1 April 2005). *Curriculum development in educating undergraduate software engineers - are students being prepared for the profession?* Paper presented at the Software Engineering Conference, 2005. Proceedings. 2005 Australian.
Sebern, M. J. (2002, 2002). *The software development laboratory: incorporating industrial practice in an academic environment.* Paper presented at the Software Engineering Education and Training, 2002. (CSEE&T 2002). Proceedings. 15th Conference on.
Shaw, M., & Tomayko, J. (1991). Models for undergraduate project courses in software engineering. In J. Tomayko (Ed.), *Software Engineering Education.* (Vol. 536, pp. 33-71): Springer Berlin Heidelberg. Retrieved from http://dx.doi.org/10.1007/BFb0024284 doi:10.1007/BFb0024284
Sherrell, L. B., & Robertson, J. J. (2006). Pair programming and agile software development: experiences in a college setting. *J. Comput. Small Coll., 22*(2), 145-153.
Simon, B., & Hanks, B. (2008). First-year students' impressions of pair programming in CS1. *Journal on Educational Resources in Computing (JERIC), 7*(4), 5.
Sims-Knight, J. E., & Upchurch, R. L. (1998, 4-7 Nov. 1998). *The acquisition of expertise in software engineering education.* Paper presented at the Frontiers in Education Conference, 1998. FIE '98. 28th Annual.
Sinderson, E., & Spirkovska, L. (2001, 2001). *Undergraduate software engineering education: the body of knowledge, existing programs and accreditation.* Paper presented at the Software Engineering Education and Training, 2001. Proceedings. 14th Conference on.
Slaten, K., Droujkova, M., Berenson, S., Williams, L., & Layman, L. (2005). *Understanding Student Perceptions of Pair Programming and Agile Software Development Methodologies: Verifying a Model of Social Interaction*. Paper presented at the Agile 2005, Denver, CO.
Snell, S., Snell-Siddle, C., & Whitehouse, D. (2002). *Soft or hard boiled: Relevance of soft skills for IS professionals.* Paper presented at the 15th Annual Conference of the National Advisory Committee on Computing Qualifications, Hamilton.
Sommerville, I. (2007). *Software Engineering* (8th ed.). Harlow: Pearson Education Limited.
Stringer, E. (2008). *Action Research in Education* (2nd ed.). Upper Saddle River: Pearson Education.
Stringer, E. (2010). Action Research in Education. In P. Peterson, E. Baker & B. McGaw (Eds.), *International Encyclopedia of Education (Third Edition).* (pp. 311-319). Oxford: Elsevier. Retrieved from http://www.sciencedirect.com/science/article/pii/B9780080448947015311

Suri, D. (2007, 10-13 Oct. 2007). *Providing "real-world" software engineering experience in an academic setting.* Paper presented at the Frontiers In Education Conference - Global Engineering: Knowledge Without Borders, Opportunities Without Passports, 2007. FIE '07. 37th Annual.

Suri, D., & Sebern, M. J. (2004, 1-3 March 2004). *Incorporating software process in an undergraduate software engineering curriculum: challenges and rewards.* Paper presented at the Software Engineering Education and Training, 2004. Proceedings. 17th Conference on.

The Standish Group International, I. (1995). *The Chaos Report (1994)*. Retrieved from http://www.standishgroup.com/sample_research/

Thompson, J. B., & Reed, K. (2005). Undergraduate software engineering education: the mark of a discipline. *Software, IEEE, 22*(6), 96-97. doi:10.1109/ms.2005.167

Upchurch, R. L., & Sims-Knight, J. E. (1998, 22-25 Feb 1998). *In support of student process improvement.* Paper presented at the Software Engineering Education, 1998. Proceedings., 11th Conference on.

Van Slyke, C., Kittner, N., & Cheney, P. (1998). Skill requirements for entry level IS graduates: A report from industry. *Journal of Information Systems Education, 9*, 7-11.

Watson, D. (2010). Soft skills lacking in candidate-rich market: survey. *Computerworld: The voice of the ICT community*. Retrieved from http://computerworld.co.nz/news.nsf/careers/soft-skills-lacking-in-candidate-rich-market-survey

Young, A., Senadheera, L., & Clear, T. (1999). *Knowledge skills and abilities demanded of graduates in the new learning environment.* Paper presented at the 12th Annual Conference of the National Advisory Committee on Computing Qualifications, Dunedin, NZ.

Appendices

Appendix One: MIT Graduate Survey (MIT project Iteration One)

Part One – General Employment Information

Name _____

Current or Last Employer _____

Position _____

1. Is the position you hold currently your first IT role? If yes, go to part two. Yes No

2. Have you worked in a software developer role previous to studying BICT? Yes No

3. List your previous IT roles and length of employment

_____

_____

Part Two – Evaluation of skills learnt from the MIT Project

For the MIT project please select which part of the development you were involved in:

Website and Database ☐

PDA Application ☐

For the following list of skills I have identified as being part of the MIT project please rate how much you learnt of this skill during the MIT project and how useful this skill has been so far in your working life

Section One: Team-based Development and Real-life clients

1. ***Agile Project Management with SCRUM***

How much did you learn about this during the MIT project?

0 Learned nothing at all	1 Became vaguely familiar	2 Learned the basics	3 Became Functional (moderate working knowledge)	4 Learned a lot	5 Learned in depth (expert)

How useful has this specific skill been to you in your career?

0 Completely Useless	1 Almost never useful	2 Occasionally useful	3 Moderately useful	4 Very useful	5 Essential

2. Peer Programming Techniques

How much did you learn about this during the MIT project?

0 Learned nothing at all	1 Became vaguely familiar	2 Learned the basics	3 Became Functional (moderate working knowledge)	4 Learned a lot	5 Learned in depth (expert)

How useful has this specific skill been to you in your career?

0 Completely Useless	1 Almost never useful	2 Occasionally useful	3 Moderately useful	4 Very useful	5 Essential

3. Requirements elicitation with real Clients

How much did you learn about this during the MIT project?

0 Learned nothing at all	1 Became vaguely familiar	2 Learned the basics	3 Became Functional (moderate working knowledge)	4 Learned a lot	5 Learned in depth (expert)

How useful has this specific skill been to you in your career?

0 Completely Useless	1 Almost never useful	2 Occasionally useful	3 Moderately useful	4 Very useful	5 Essential

4. General Team Dynamics (communication, priority negotiation, conflict resolution amongst team members)

How much did you learn about this during the MIT project?

0 Learned nothing at all	1 Became vaguely familiar	2 Learned the basics	3 Became Functional (moderate working knowledge)	4 Learned a lot	5 Learned in depth (expert)

How useful has this specific skill been to you in your career?

0 Completely Useless	1 Almost never useful	2 Occasionally useful	3 Moderately useful	4 Very useful	5 Essential

Section Two: Technical Skills Developed

For this section:

If you were part of the Web Development Team please answer questions 5, 6 and 9.

If you were part of the PDA application Team please answer questions 7, 8 and 9.

5. ASP.Net Web development

How much did you learn about this during the MIT project?

0 Learned nothing at all	1 Became vaguely familiar	2 Learned the basics	3 Became Functional (moderate working knowledge)	4 Learned a lot	5 Learned in depth (expert)

How useful has this specific skill been to you in your career?

0 Completely Useless	1 Almost never useful	2 Occasionally useful	3 Moderately useful	4 Very useful	5 Essential

6. SQL Server (Data structure and SQL queries)

How much did you learn about this during the MIT project?

0 Learned nothing at all	1 Became vaguely familiar	2 Learned the basics	3 Became Functional (moderate working knowledge)	4 Learned a lot	5 Learned in depth (expert)

How useful has this specific skill been to you in your career?

0 Completely Useless	1 Almost never useful	2 Occasionally useful	3 Moderately useful	4 Very useful	5 Essential

7. C#. Net and the .Net Compact Framework

How much did you learn about this during the MIT project?

0 Learned nothing at all	1 Became vaguely familiar	2 Learned the basics	3 Became Functional (moderate working knowledge)	4 Learned a lot	5 Learned in depth (expert)

How useful has this specific skill been to you in your career?

0 Completely Useless	1 Almost never useful	2 Occasionally useful	3 Moderately useful	4 Very useful	5 Essential

8. RDA data transfer and the compact edition database

How much did you learn about this during the MIT project?

0 Learned nothing at all	1 Became vaguely familiar	2 Learned the basics	3 Became Functional (moderate working knowledge)	4 Learned a lot	5 Learned in depth (expert)

How useful has this specific skill been to you in your career?

0 Completely Useless	1 Almost never useful	2 Occasionally useful	3 Moderately useful	4 Very useful	5 Essential

9. Application Security (such as: privileges, user authorisation, encryption)

How much did you learn about this during the MIT project?

0 Learned nothing at all	1 Became vaguely familiar	2 Learned the basics	3 Became Functional (moderate working knowledge)	4 Learned a lot	5 Learned in depth (expert)

How useful has this specific skill been to you in your career?

0 Completely Useless	1 Almost never useful	2 Occasionally useful	3 Moderately useful	4 Very useful	5 Essential

Part Three - Real-life Project Experience

10. Did your confidence in yourself as an analyst / developer improve during the project?

Yes ☐ No ☐

11. Did you refer to the project as development experience in your CV?

Yes ☐ No ☐

12. Did you use the project as an example of development experience in your job interview?

Yes ☐ No ☐

a. If you answered yes above, which aspects of the project did you refer to?

Teamwork experience ☐

Experience in a particular technology ☐

Other _____ (please specify)

Part Four - Skill Gaps

13. If there were any skills that were required of you when you began your first ICT role that you had no knowledge of and that you feel could be taught in the classroom please list them here:

_____

_____

Thank you for your time, your input is valued

Appendix Two: MIT Graduate Survey (MIT project Iteration Two)

Part One – General Employment Information

Name _____

Employer _____

Position _____

1. Is the position you hold currently your first IT role? If yes, go to Part two. Yes No

2. Have you worked in a software developer role previous to studying BICT? Yes No

3. List your previous IT roles and length of employment

_____
_____
_____
_____

Part Two – Evaluation of skills learnt from the MIT Project

For the following list of skills I have identified as being part of the MIT project please rate how much you learnt of this skill during the MIT project and how useful this skill has been so far in your working life

Section One: Team-based Communication and Tools

1. ***Visual Source Safe – Code Repository and Version Control***

How much did you learn about this during the MIT project?

0 Learned nothing at all	1 Became vaguely familiar	2 Learned the basics	3 Became Functional (moderate working knowledge)	4 Learned a lot	5 Learned in depth (expert)

How useful has this specific skill been to you in your career?

0 Completely Useless	1 Almost never useful	2 Occasionally useful	3 Moderately useful	4 Very useful	5 Essential

2. Team-based communication using discussion forums

How much did you learn about this during the MIT project?

0 Learned nothing at all	1 Became vaguely familiar	2 Learned the basics	3 Became Functional (moderate working knowledge)	4 Learned a lot	5 Learned in depth (expert)

How useful has this specific skill been to you in your career?

0 Completely Useless	1 Almost never useful	2 Occasionally useful	3 Moderately useful	4 Very useful	5 Essential

3. Working with a large existing codebase (i.e. getting your head around someone else's code)

How much did you learn about this during the MIT project?

0 Learned nothing at all	1 Became vaguely familiar	2 Learned the basics	3 Became Functional (moderate working knowledge)	4 Learned a lot	5 Learned in depth (expert)

How useful has this specific skill been to you in your career?

0 Completely Useless	1 Almost never useful	2 Occasionally useful	3 Moderately useful	4 Very useful	5 Essential

4. General Team Dynamics (communication, priority negotiation, conflict resolution amongst team members)

How much did you learn about this during the MIT project?

0 Learned nothing at all	1 Became vaguely familiar	2 Learned the basics	3 Became Functional (moderate working knowledge)	4 Learned a lot	5 Learned in depth (expert)

How useful has this specific skill been to you in your career?

0 Completely Useless	1 Almost never useful	2 Occasionally useful	3 Moderately useful	4 Very useful	5 Essential

Section Two: Technical Skills Developed

5. C#. Net and the .Net Compact Framework

How much did you learn about this during the MIT project?

0 Learned nothing at all	1 Became vaguely familiar	2 Learned the basics	3 Became Functional (moderate working knowledge)	4 Learned a lot	5 Learned in depth (expert)

How useful has this specific skill been to you in your career?

0 Completely Useless	1 Almost never useful	2 Occasionally useful	3 Moderately useful	4 Very useful	5 Essential

6. RDA transfer and the compact edition database

How much did you learn about this during the MIT project?

0 Learned nothing at all	1 Became vaguely familiar	2 Learned the basics	3 Became Functional (moderate working knowledge)	4 Learned a lot	5 Learned in depth (expert)

How useful has this specific skill been to you in your career?

0 Completely Useless	1 Almost never useful	2 Occasionally useful	3 Moderately useful	4 Very useful	5 Essential

7. SQL Server (Data structure and SQL queries)

How much did you learn about this during the MIT project?

0 Learned nothing at all	1 Became vaguely familiar	2 Learned the basics	3 Became Functional (moderate working knowledge)	4 Learned a lot	5 Learned in depth (expert)

How useful has this specific skill been to you in your career?

0 Completely Useless	1 Almost never useful	2 Occasionally useful	3 Moderately useful	4 Very useful	5 Essential

8. ASP.Net Web development

How much did you learn about this during the MIT project?

0 Learned nothing at all	1 Became vaguely familiar	2 Learned the basics	3 Became Functional (moderate working knowledge)	4 Learned a lot	5 Learned in depth (expert)

How useful has this specific skill been to you in your career?

0 Completely Useless	1 Almost never useful	2 Occasionally useful	3 Moderately useful	4 Very useful	5 Essential

9. Application Security (such as: privileges, user authorisation, encryption)

How much did you learn about this during the MIT project?

0 Learned nothing at all	1 Became vaguely familiar	2 Learned the basics	3 Became Functional (moderate working knowledge)	4 Learned a lot	5 Learned in depth (expert)

How useful has this specific skill been to you in your career?

0 Completely Useless	1 Almost never useful	2 Occasionally useful	3 Moderately useful	4 Very useful	5 Essential

Part Three - Real-life Project Experience

10. Did your confidence in yourself as a developer improve during the project?

Yes ☐ No ☐

11. Did you refer to the project as development experience in your CV?

Yes ☐ No ☐

12. Did you use the project as an example of development experience in your job interview?

Yes ☐ No ☐

a. If you answered yes above, which aspects of the project did you refer to?

Teamwork experience ☐

Experience in a particular technology ☐

Other _____ (please specify)

Part Four - Skill Gaps

13. If there were any skills that were required of you when you began your first ICT role that you had no knowledge of and that you feel could be taught in the classroom please list them here:

_____

_____

Thank you for your time, your input is valued

Appendix Three : Employer Survey

Organisation Name:

Number of IT employees:

Your Position:

For the following list of technical and soft skills, please rate how important you think this skill is for your graduate IT workers to possess (1= Unimportant, 5= Very Important)

T**echnical Skills**

Category	**Sub Category**		**1 Unimportant**	**2 Of Little Importance**	**3 Moderately important**	**4 Important**	**5 Very Important**
OO Programming	C# .Net						
	VB .Net						
	Java						
	Other (please specify) _____						
3GL Procedural Programming	Delphi/ Pascal						
	Visual Basic						
	Other (please specify) _____						
Email/Chat/Office Communicator							
Structured Analysis & Design	e.g. DFD / ERD						
OO Analysis & Design	e.g. UML						
Database Administration	MS SQL Server						
	Oracle						
	MS Access						
	Other (please specify) _____						
Team Development Tools	Source Repository						
	Team portal / forum						
Project Management	PMBOK						
	Agile Methods						
Operating Systems	XP						
	Windows 7						
Internet /Ecommerce	ASP .NET						
	PHP						
	JSP						
	HTML & JavaScript						
Networking & Network Administration							
Mobile Application Development	.NET Compact Framework						
	J2ME						
Data Warehousing							
User Training and Support							
Use & evaluation of software packages							

Soft Skills

	1 Unimportant	2 Of Little Importance	3 Moderately important	4 Important	5 Very Important
Communication					
Creative Thinking					
Global mindset					
Initiative & Enterprise					
Problem solving					
Information gathering techniques					
Self management					
Team working skills					

Finally, are there any skills you believe are critical that I have not covered in the skill lists above?

Please list below:

_____

_____

Thank you very much for your participation, your input is valued.

Appendix Four: Information and Consent Forms

Information Form – Employer

Curtin University

Science and Mathematics Education Centre

Participant Information Sheet

My name is Sandra Cleland. I am currently completing a piece of research for my Masters of Philosophy (Science Education) at Curtin University.

Purpose of Research

I am investigating how effective a real-life, software development project undertaken by third-year software engineering students was at providing them with the skills that were required in their first analyst / developer role

Your Role

I am interested in finding out what skills you believe are the most important ones that your graduate ICT employees should possess.
The questionnaire will take approximately 10 minutes to complete

Consent to Participate

Your involvement in the research is entirely voluntary. You have the right to withdraw at any stage without it affecting your rights or my responsibilities. When you have returned the consent form via email I will forward the questionnaire and will use your response as data in this research

Confidentiality

The information you provide will be kept separate from your personal details, and only myself and my supervisor will only have access to this. The questionnaire data will not have your name or any other identifying information on it and in adherence to university policy, the data will be kept in a locked cabinet for five years, before it is destroyed.

Further Information

This research has been reviewed and given approval by Curtin University Human Research Ethics Committee (Approval Number *SMEC-61-10*). If you would like further information about the study, please feel free to contact me by email: s.cleland@ucol.ac.nz or by phone +64 6 9527001 ext 70120. Alternatively, you can contact my supervisor Professor Darrell Fisher on +61 8 92663110 or email: D.Fisher@curtin.edu.au.

Thank you very much for your involvement in this research.

Your participation is greatly appreciated.

Information Form – Graduate Participants

Curtin University

Science and Mathematics Education Centre

Participant Information Sheet

My name is Sandra Cleland. I am currently completing a piece of research for my Masters of Philosophy (Science Education) at Curtin University.

Purpose of Research

I am investigating how effective the real-life, software development project we undertook for the Medical Imaging Technology students was at providing you with skills that were required in your first ICT role.

Your Role

I am interested in finding out what skills you believe were the most important ones developed during this project. Secondly, I would like to find out if there are any skill areas where you felt your education had done little to prepare you for first role in ICT. The questionnaire will take approximately 15 minutes to complete

Consent to Participate

Your involvement in the research is entirely voluntary. You have the right to withdraw at any stage without it affecting your rights or my responsibilities. When you have returned the consent form via email I will forward the questionnaire and will use your response as data in this research

Confidentiality

The information you provide will be kept separate from your personal details, and only myself and my supervisor will only have access to this. The questionnaire data will not have your name or any other identifying information on it and in adherence to university policy, the data will be kept in a locked cabinet for five years, before it is destroyed.

Further Information

This research has been reviewed and given approval by Curtin University Human Research Ethics Committee (Approval Number *SMEC-61-10*). If you would like further information about the study, please feel free to contact me by email: s.cleland@ucol.ac.nz or by phone +64 6 9527001 ext 70120. Alternatively, you can contact my supervisor Professor Darrell Fisher on +61 8 92663110 or email: D.Fisher@curtin.edu.au.

Thank you very much for your involvement in this research. Your participation is greatly appreciated.

Information Form – Student Participants

Curtin University

Science and Mathematics Education Centre

Participant Information Sheet

My name is Sandra Cleland. I am currently completing a piece of research for my Masters of Philosophy (Science Education) at Curtin University.

Purpose of Research

I am investigating how effective the real-life, software development project undertaken in D301 is at providing graduates with the skills that they require in their first ICT role.

Your Role

During this semester in D301 you will be observed in the classroom participating in the team development project. I will use these observations alongside your submitted reflective journals to identify challenges you faced during the semester.

Consent to Participate

Your involvement in the research is entirely voluntary. You have the right to withdraw at any stage without it affecting your rights or my responsibilities. When you have graduated and secured your first ICT position you will be provided with a questionnaire and I will use your response as data in this research

Confidentiality

The information you provide will be kept separate from your personal details, and only myself and my supervisor will only have access to this. The observational data will not have your name or any other identifying information on it and in adherence to university policy, the data will be kept in a locked cabinet for five years, before it is destroyed.

Further Information

This research has been reviewed and given approval by Curtin University Human Research Ethics Committee (Approval Number *SMEC-61-10*). If you would like further information about the study, please feel free to contact me by email: s.cleland@ucol.ac.nz or by phone +64 6 9527001 ext 70120. Alternatively, you can contact my supervisor Professor Darrell Fisher on +61 8 92663110 or email: D.Fisher@curtin.edu.au.

Thank you very much for your involvement in this research. Your participation is greatly appreciated.

Consent Form

CONSENT FORM

_____

• I understand the purpose and procedures of the study.

• I have been provided with the participation information sheet.

• I understand that the procedure itself may not benefit me.

• I understand that my involvement is voluntary and I can withdraw at any time without problem.

• I understand that no personal identifying information like my name and address will be used in any published materials.

• I understand that all information will be securely stored for at least 5 years before a decision is made as to whether it should be destroyed.

• I have been given the opportunity to ask questions about this research.

• I agree to participate in the study outlined to me.

_____

Name: _____

Signature: _____

Date: _____