
© Copyright 2010 Australian Computer Society, Inc.
Engaging Students in Programming
Malcolm Corney, Donna Teague and Richard N. Thomas
Faculty of Science and Technology
Queensland University of Technology
Brisbane, Australia

[m.corney, d.teague, r.thomas]@qut.edu.au

Abstract
Poor student engagement and high failure rates in first year units were addressed at the Queensland University of Technology (QUT) with a course restructure involving a fresh approach to introducing programming. Students’ first taste of programming in the new course focused less on the language and syntax, and more on problem solving and design, and the role of programming in relation to other technologies they are likely to encounter in their studies. In effect, several technologies that have historically been compartmentalised and taught in isolation have been brought together as a breadth-first introduction to IT.

Incorporating databases and Web development technologies into what used to be a purely programming unit gave students a very short introduction to each technology, with programming acting as the glue between each of them. As a result, students not only had a clearer understanding of the application of programming in the real world, but were able to determine their preference or otherwise for each of the technologies introduced, which will help them when the time comes for choosing a course major.

Students engaged well in an intensely collaborative learning environment for this unit which was designed to both support the needs of students and meet industry expectations. Attrition from the unit was low, with computer laboratory practical attendance rates for the first time remaining high throughout semester, and the failure rate falling to a single figure percentage.

Keywords: introductory programming, IT course, student engagement, attrition

1 Introduction
Attrition from programming courses is historically high (Berenson, Slaten et al. 2004; Kinnunen and Malmi 2006; Biggers, Brauer et al. 2008), particularly in minority groups for whom there is often poor representation to begin with (Cohoon 2002; Fisher and Margolis 2002; Lewis, McKay et al. 2006; Varma 2006; Vilner and Zur 2006).

Introductory programming units in particular have had an alarming failure rate (Sheard and Hagan 1998; Robins, Rountree et al. 2003). More than 30% of QUT students on average had failed QUT’s introductory programming subject since 2003, and for some semesters that percentage was significantly higher (Teague and Roe 2009).

Much research has focused on this dilemma culminating in a range of cognitive theories for high failure rates including:
- the difficulty of understanding the purpose of programs and their relationship with the computer;
- difficulty in grasping the syntax and semantics of a particular programming language (Robins, Rountree et al. 2003);
- misconceptions of programming constructs (Soloway and Spohrer 1989);
- inability to problem-solve (McCracken, Almstrom et al. 2001); and
- inability to read and understand program code (Lister, Adams et al. 2004; Mannila 2006).

The idea that the failure lies in the ability of some students to grasp the course content has seen repeated redevelopment of introductory programming courses with changes of language, paradigm, and swapping between breadth and depth of content approaches.

Motivation, however, has been found to be one of the major reasons for students dropping out of IT courses (Kinnunen and Malmi 2006). QUT’s data shows that there is a correlation between students who fail an introductory programming unit and those who withdraw from their degree. This has been informally confirmed by many other institutions. The implicit assumption is that lack of motivation leads to failure or that poor performance in early assessment tasks leads to poor motivation.

To address student motivation by making learning more fun, courses and tools have been developed to help captivate introductory programming students (Lister 2004; Parsons and Haden 2006; Pollard and Duvall 2006; Davis and Rebelsky 2007; Feinberg 2007).

The ‘big picture’ for many students is that programming is perceived as a solitary occupation, and one which is conducted in a competitive environment. This is unwittingly reinforced at university where faculties demand that programming students be individually assessed. The generally accepted stereotype of a programmer is the ‘geeky’ young male with dubious social skills, and it is not surprising that many students are alienated by this image. This may often negatively affect their confidence and lead to a subsequent lack of engagement and interest (Fisher and Margolis 2002).

Collaborative learning establishes an environment conducive to learning and addresses some of the social and cultural barriers facing students (Wilson, Hoskin et al. 1993; Williams and Kessler 2000; McDowell, Werner et al. 2002; Gehringer, Deibel et al. 2006). It has been
found that students benefit from the peer support while learning, and at the same time are motivated by peer pressure and a sense of purpose and belonging (McKinney and Denton 2006).

Taking the collaborative approach by using pair programming in the learning environment has been documented as having significant educational benefits including active learning and improved retention, program quality, and confidence in the solution (McDowell, Werner et al. 2002; Nagappan, Williams et al. 2003; McDowell, Werner et al. 2006; Mendes, Al-Fakhri et al. 2006). A more social rather than competitive environment is established with pair programming which promotes more interaction and lends twice as much brain power and an extra set of eyes to a programming exercise (Simon and Hanks 2007).

In this paper we report on our approach to redesigning the introductory programming unit at QUT. Our aim was to focus on the issues of motivation and engagement, as well as the social and cultural attitudes affecting the way students engage in their learning to program. We provided students with an intensely collaborative learning environment and were better able to engage students in programming by providing them with a taste of a number of technologies, each of which relies on programming.

The redesign of this introductory unit came about within a new overall course structure for the Bachelor of IT. By providing students with introductions to various technologies, it was hoped that students with a preference for a particular area of study could get a brief taste of that area before embarking on a more serious investment of time in a particular unit.

The remainder of the paper is structured as follows. In Section 2, we discuss previous approaches taken in teaching introductory programming at QUT and the perceived problems with those approaches. Section 3 presents the approach we have developed for the new introductory programming unit, describing the unit’s structure and how it fits into the new course structure, and the unit’s approach to teaching and assessment. An analysis of the student cohort enrolled in the unit is also provided. The evidence of our success in improving the unit is presented in Section 4. Finally we present our future plans for the unit and our conclusions in Sections 5 and 6.

2 Historical Units

2.1 Introductory Programming Units

The teaching of programming at QUT has undergone an evolution over the past twenty years as undergraduate degree course structures were reviewed and redeveloped. Introductory programming has always been seen as a requirement for all students enrolled in our IT course by industry advisers. QUT has recently offered an undergraduate degree program in Games and Interactive Entertainment (GIE) and advisers in this field similarly agree that programming is necessary.

Over the past eight years many changes were made to our introductory programming units as the number of students entering IT degree courses first swelled and then dwindled for many reasons – the rise and fall of the tech bubble, post Y2K, programming and programmers being seen as an outsourced commodity etc.

Rising numbers of students lead to problems in the quality of teaching and learning inherent with large class sizes and the overall student experience suffered accordingly.

A subsequent and dramatic decline in the number of students entering the course resulted in fewer students with high university entrance scores which affected the nature of the student cohort. Methods of teaching which had previously been seen as successful were no longer working as well as they had.

Student cohorts typically contained a mixture of students who could already program to some extent and those who had never programmed before. As the introductory units were aimed at the lowest common denominator, students with programming skills typically saw the units as too simple and failed to engage with the content. Other students simply did not cope with learning to program for any number of reasons, and then failed to engage as they fell behind in their learning.

Assessment in these units typically consisted of one or two programming assignments of varying levels of difficulty and an end of semester exam which was worth 70% of their final grade for the unit. Assignments were often quite large in the context of novice programming, were often poorly attempted and were rarely completed successfully. While formative feedback was provided to students for assignments, it was typically not useful to the students in examinations.

Recent offerings of first programming units attempted to focus on problem solving but this was done with an emphasis on producing programs, rather than through categorization of problems into types that can be solved with different algorithmic approaches.

2.2 Approaches Taken

In previous course designs a number of approaches have been taken in the teaching of introductory programming units at QUT. Imperative programming was used initially with languages such as Pascal and Modula 2. Attempts were made to introduce object oriented programming with Java and C#, though this was mainly taught with an introduction to imperative programming before objects were introduced. A purely functional approach was also trialled using Scheme as the language du jour.

2.3 The Impact of Course Structure

The design of many information technology courses has compartmentalized the teaching of the main building blocks of IT systems development, i.e. programming, database design, administration and use; and Web development. Each of these areas has been taught as a separate body of knowledge that IT students should learn in isolation.

The IT course at QUT had been altered after faculty reviews to address falling student numbers. However, little was done in these reviews to address the poor student outcomes for introductory programming.
2.4 Retention

Retention of students in introductory programming units suffered as the student cohort altered as discussed above. This was evidenced each semester at census dates by shrinking class sizes.

Tutorial and/or practical attendances nearly always declined as the weeks of the semester progressed. Factors such as workload from other units and external commitments were often blamed but a major factor may likely have been lack of engagement with the unit materials.

Overall results for introductory programming units in the past have typically had a bimodal distribution with one mode for those students who either had prior knowledge of programming or managed to learn it quickly and another mode for those students who did not manage to learn to program well at all.

Students who performed poorly in these programming units, which were normally core to all IT courses, were unlikely to continue tertiary study and those who were enrolled in double degrees would often discontinue the IT degree. As Table 1 shows, semester 1, 2008 saw 19% of students failing the first programming unit. In the same semester, 35% of students discontinued their course with over half of them withdrawing from university altogether (see Table 2).

<table>
<thead>
<tr>
<th>Semester 1, 2008</th>
<th>First Programming Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>pass</td>
<td>81%</td>
</tr>
<tr>
<td>fail</td>
<td>19%</td>
</tr>
</tbody>
</table>

Table 1: Fail/Pass Rates

<table>
<thead>
<tr>
<th>Semester 1, 2008</th>
<th>Attrition Rates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Changed to other course</td>
<td>16.2%</td>
</tr>
<tr>
<td>inactive/on leave</td>
<td></td>
</tr>
<tr>
<td>Discontinued course enrolment</td>
<td>18.4%</td>
</tr>
<tr>
<td>withdrew from First Programming Unit</td>
<td>19.4%</td>
</tr>
</tbody>
</table>

Table 2: Attrition Rates

3 The New Course Structure

The design goal for the first semester core of the new Bachelor of IT (BIT), introduced in 2009, was to improve student engagement, and consequently progression, while maintaining the quality of our graduates. The core maintains the idea of ensuring that all of our BIT graduates have a common set of skills and knowledge.

The technical material in the previous core was taught pretty much the same way for the past 30 years. Many of the technical units had poor progression rates and particularly the programming and database units had very poor progression rates. The Faculty of Science and Technology (SciTech’s) approach to teaching the technical material was similar to most other mainstream IT degrees and we had similarly poor progression rates. We needed a different approach to introducing technical content to see significant changes in progression rates.

The design of the new BIT took into account our experiences teaching IT, model curricula and research in pedagogy. In particular we considered the guidelines provided by the ACM, AIS and IEEE Computer Society Computing Curricula 2005 (The Joint Task Force for Computing Curricula ACM/AIS/IEEE-CS 2006) along with its companion volumes Computer Science (The Interim Review Task Force ACM/IEEE-CS 2008), Information Systems (Gorgone, Davis et al. 2002), Information Technology (Lunt, Ekstrom et al. 2008) and Software Engineering (The Joint Task Force on Computing Curricula IEEE-CS/ACM 2004). (In the remainder of this section we will use the term CC2005 to refer to Computing Curricula 2005 and all of its companion volumes.) CC2005 is a content driven view of curricula. The design of the new BIT took into account the knowledge areas and recommended topic weighting from CC2005 but we purposely did not aim to meet the recommended weightings. Given the breadth of computing and the need to adequately develop generic capabilities we found the CC2005 recommendations to be too heavily biased towards content over general abilities. We also noted that even the content recommended by CC2005 was too narrow to adequately cover both the breadth of information technology and the associated non-computing knowledge required by graduates. A recent workshop on redefining computing curricula noted that in a short period of time they were able to identify over 100 additional knowledge areas relevant to computing that were not covered by CC2005 (Isbell, Stein et al. 2010). In the end, the design of the new BIT followed current pedagogical research being undertaken at QUT (QUT 2009). The first semester core units and units which immediately follow focus on preparing students for their university studies. The middle part of the degree focuses on delivering relevant material in realistic contexts. The final year core units focus on preparing students for their post-degree careers with an emphasis on integrating the topics studied earlier in their degree and on understanding the context in which they will apply the knowledge and skills.

Consequently the new core has a shallower introduction to the technical content and introduces this content using an integrated approach, which matches how the content is used in practice. We wanted to shift the focus from didactic teaching to a more constructivist approach to learning (Bowden and Marton 1999). Our own experience introducing Problem-Based Learning (PBL) into programming units (Adams, Clarke et al. 2001) and the experience of MacDonald (1998) in medicine validates this shallower introduction of material. The experience from the PBL community is that a limited and focussed coverage of content provides a better basis for students to learn material on their own, rather than trying to cram everything into a course. The integrated approach to introducing content material was informed by the framework for Teaching and Assessment of Software Development (Thomas, Cordiner et al. 2010), which provides a structure for designing a stream of technical units that make up a coherent whole. We believe that if students are engaged in the learning process and the material, they will learn the fundamental concepts well giving them adequate skills for the remainder of their degree, regardless of the area in which they specialise.

The core is intended to develop well rounded students, and leave technical depth to a set of specialisation
streams. This reflects industry feedback on requirements for graduates with better business, communication and teamwork skills, and also on the expected technical ability of our graduates. The first semester units are not meant to eliminate difficult technical material, but to present it in a more engaging manner which better shows how the course content inter-relates. The motivation for this is the number of our students who do not see how the different aspects of IT fit together (e.g. believing that working with databases does not require programming skills).

In student focus groups conducted at QUT, some of the comments from students were that they wanted more "hands on" work, more exposure to "real world" projects and examples, a better introduction to object-oriented programming, and more industry certification options. Double degree students commented that SciTech was better at building student cohorts than other faculties and that we were better at getting students to work together. Most students disliked working in teams if some team members did not "pull their weight". However, we recognise the importance of students building a network of friends during their transition to university, as well as the valuable generic skills that collaborative learning affords them.

INB104 – Building IT Systems – is one of four core units offered in the first semester of the first year of study for Bachelor of IT students. It is also a core unit for Bachelor of Games and Interactive Entertainment students and for all students enrolled in double degrees involving either of these degrees. Double degree students typically undertake study in INB104 in the second semester of their course.

The four core units form a coherent group which expose students to: the breadth of domains in which IT is used and how IT has changed those domains; advances in computing devices to introduce hardware architecture, networking and operating systems; and an introduction to the profession of IT and the generic skills required by all IT graduates.

INB104 rounds this set of units out with an introduction to the basic building blocks of IT systems: networks, databases, and software – programs, scripts and Web development. While this may seem like a large amount of material, the topics are covered without going into too great a depth.

An aim of the unit is to provide students with some experience in these building blocks before they choose a set of follow-on units in second semester, which includes Programming, Databases, Networks, and the Web. These units then lead on to the areas in which students may specialise, such as enterprise systems, software engineering, Web development or networking.

3.1 Introducing INB104 – Building IT Systems

Combining programming, databases and Web development into one first year unit allows the students to gain an earlier understanding of these basic concepts albeit at a more general level. The main aim of this unit is to engage the students in these building blocks and to learn the basics by immersing them in a variety of interesting tasks that will use one, two or all three of the technologies. Programming is being used as the glue between the systems, rather than simply for the sake of something that must be learnt.

Given the range of topics in INB104, programming skills are restricted to the fundamental concepts of sequence, selection, iteration and functions. We decided to use Python as the introductory language due to our experience with the language and its simplicity. Python is open source, involves hassle-free installation and has a simple syntax and development interface. It can be used for writing simple scripts, full blown programs and for the creation of object-oriented systems. It is also possible to use the functional programming paradigm with this language.

INB104 focuses on imperative programming using a top down design approach which motivates the use of functions. An imperative approach was chosen because it provides the best approach to teaching the fundamental concepts covered in the unit. A functional or object-oriented approach would require too much overhead in the form of other concepts to suit the goals of the unit when using Python. It is also an approach that suits the design of scripting programs as well as providing the underlying algorithmic logic required by object-oriented programming. Students will encounter both of these approaches in following units. Python library modules are utilised for animation (PyGame), image manipulation (PIL), and database communication (MySQLDb).

MySQL is used for projects which require database interaction. It provides good GUI based tools for interaction with a database server, so that SQL queries can be tested before they are used in Python programs.

The Web development element of the unit covers a subset of HTML and uses simple text editors in preference for sophisticated development environments like DreamWeaver.

INB104 highlights the fact that SQL and HTML are IT system development languages that can be used to build interactive systems.

3.2 Teaching Approach

Students are expected to attend a two hour demonstration-style lecture and a two hour computer laboratory based tutorial session each week. The laboratory sessions are supervised by one tutor for each class of 25 to 30 students. This level of support has been provided to our first year students for some time now and does not represent any increase in resources required for delivery of the unit.

Lecture slides and laboratory worksheets are published online prior to delivery and audio recordings are made available following the lectures.

In the first week of semester, students are encouraged to partake in an animated social discussion in the practical sessions, aimed at breaking the ice and getting to know a little about their fellow students. Students then self-select into pairs. They are asked to choose carefully, a partner they believe they will be able to work with effectively throughout semester. Students are encouraged to consider such things as demographics, culture, background, motivation, programming experience and timetabling. They are then introduced to the concept and protocol of pair programming (Beck 2005) and provided
with background information supporting its use in both industry and the learning environment. In the university environment, we believe that the student pairs should be fixed for the semester as the main aim of the learning experience is the content matter rather than the pair programming protocol. In this and previous pair programming experiments (Teague and Roe 2009) we have found that after an initial settling in period, students prefer to continue with the same partner for the duration of the semester rather than spend time developing a workable relationship with someone new.

Paired students are encouraged to actively engage with their partner to complete practical exercises both during practicals and at other negotiated times either on or off campus. Tutors enforce regular swapping of roles between driver and navigator, reminding them of ways they can be active members of the pair including asking questions, offering alternatives, researching syntax etc.

Activities in the weekly computer based practical sessions are all directly related to the unit content delivered at the prior lecture and these activities in many cases feed directly into similar tasks that are required for assessment. This approach is seen by the unit developers as vital for providing relevance to the weekly schedule and engagement with the unit material.

3.3 Assessment

Previous studies (McDowell, Werner et al. 2002; Urness 2009) have found that assignments developed by novice programmers involving pair-programming result in better quality code than individual submissions. The same studies also note that similar exam averages are achieved by those learning in a pair-programming environment as those who have worked individually throughout semester.

Assessment for this unit is a combination of a portfolio of activities undertaken during the semester, a reflective report comparing initial and final skills in the areas taught in the unit, and weekly online quizzes.

‘Threshold concepts’ that novice programmers encounter and often get stuck on (Eckerdal, McCartney et al. 2006) are made less of an issue with the support of collaborative learning, and by offering a range of assessment items that appeal to the students’ sense of enjoyment and target their interest areas e.g. gaming, graphics, and Web development.

The assessment items are described in more detail below.

3.3.1 Portfolio

Small projects are developed by pairs but teamwork is not assessed. Student pairs spend some time working on the project tasks in computer laboratory practical sessions but are expected to complete the tasks using the same collaborative protocols at negotiated times and places with their partner between practicals. The students use a problem solving framework which provides a scaffold for developing the required skills. Students are expected to take an inventory of their current skills at the beginning of the task and then determine an approach that will enable them to complete the task.

The tasks are somewhat open ended in their definition and most include challenge content, allowing those students who have pre-existing skills to use and challenge those skills. For those students with little or no prior knowledge, supporting material is supplied in lectures and readings, while laboratory exercises supported by tutors provide the opportunity for them to develop the required skills with hands-on practice and experimentation. For the first offering of this new unit, the portfolio was submitted twice in draft form, allowing the students to receive feedback on their submissions before finally being graded. The drafts were due for submission at the end of weeks four and eight, with the final portfolio submitted at the end of week 13.

Each submission required a selection of two projects per pair, resulting in a final portfolio containing six projects overall. The first draft submission required a choice of practical exercises which students had been completing in class. These were programming exercises which entailed writing simple functions with or without parameters which involved sequential expressions, Boolean logic, conditions, iteration and/or the use of Turtle graphics.

The second draft submission of the portfolio included a selection of more challenging programming projects, some of which required a little research and development of skills not covered in class. One example of this was a project to produce a bouncing balls animation using an imported PyGame library. Other projects available for selection at this stage involved using Lindemayer systems (Prusinkiewicz and Lindenmayer 1990) for drawing fractal patterns with Turtle graphics and a language translator using Python’s built-in dictionary data type and its functionality.

The final portfolio stage offered the choice of projects which involved programming, SQL and/or HTML development. A programming and SQL project required students to populate an existing database with the contents of supplied text files. One project required the use of all three technologies, in which students produced a HTML popularity cloud of gathered student data. Another project had students design a set of static Web pages to display images captured by traffic cameras deployed around the South East corner of Queensland in the form of a traffic camera wall.

3.3.2 Reflective Report

A brief search indicates that few programming courses incorporate the use of reflective practice by its students. Zagal and Bruckman (2007) talk about a blogging environment developed for students as a learning tool, to reflect on their game-play experiences. Kay et al (2007) incorporated student reflective practice in their programming education system which was designed to facilitate self-assessment and reflection.

The reflective report for this unit required the students to compare their initial and final skill levels in the areas of computer programming, database usage and Web programming. The students were instructed to undertake a learning style survey (Fleming 2009) and to reflect upon how that style was manifested during their learning in this unit.

The development of the report was guided by a series of questions that asked about knowledge of a topic, comparison of understanding between the beginning and
end of the semester and whether or not the student enjoyed learning about the topic. There was also a further question asking if the student would enrol in further units related to the topic.

3.3.3 Quizzes
For the first ten weeks of the unit students were required to complete an on-line quiz, each contributing 1% towards their final grade. The first quiz was a questionnaire designed simply to gather information about students’ IT and programming skills as well as perceptions and attitudes which was later used by them to help reflect on their development of skills and knowledge throughout the semester.

The remainder of the quizzes consisted of multiple choice questions which tested their knowledge of concepts covered in the previous week’s lecture and practical session. Only one attempt per student was permitted for each quiz, and they were expected to complete the quizzes in their own time, within a week or so from being available online. With no real time restriction for completing a quiz and the freedom to use whatever resources they felt necessary in order to answer the questions, each quiz provided the opportunity for students to reflect on their understanding of technical content.

3.4 Cohort Analysis
The cohort for this unit’s first offering did not consist entirely of straight Information Technology students. Many of its students were enrolled in the School of IT’s new Bachelor of Games and Interactive Entertainment (BGIE) degree and many combine their studies in IT with a second degree. Information on the cohort is reported below.

We have also surveyed the learning style preferences of students in the cohort, using an on-line survey (Fleming 2009). Information gathered from this survey will be used to ensure that all learning styles are catered to in future offerings of the unit. The predominant learning styles of the cohort are also reported below.

3.4.1 Course Breakdown
The students undertaking the course come mainly from the School of IT’s two main courses, the Bachelor of Information Technology (BIT) and Bachelor of Games and Interactive Entertainment (BGIE) and double degrees paired with these. Table 3 shows the breakdown of courses in which students undertaking this unit are enrolled. Further information about the student cohort undertaking the unit is shown in Table 4.

<table>
<thead>
<tr>
<th>Course</th>
<th>Semester 1</th>
<th>Semester 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT</td>
<td>34%</td>
<td>26%</td>
</tr>
<tr>
<td>BIT/Double</td>
<td>19%</td>
<td>42%</td>
</tr>
<tr>
<td>BGIE</td>
<td>39%</td>
<td>4%</td>
</tr>
<tr>
<td>BGIE/Double</td>
<td>1%</td>
<td>18%</td>
</tr>
<tr>
<td>Other Single Degree</td>
<td>4%</td>
<td>8%</td>
</tr>
<tr>
<td>Other Double Degree</td>
<td>1%</td>
<td>1%</td>
</tr>
<tr>
<td>Other</td>
<td>3%</td>
<td>2%</td>
</tr>
</tbody>
</table>

Table 3: Breakdown of Degree Course Enrolments

4 Measures of Success
Our approach in measuring the evidence of the success of the unit was as follows:
- Comparison of student results before and after the introduction of this unit.
- Comparison of student attrition rates before and after the introduction of this unit.
- Comparison of number of instances of plagiarism before and after the introduction of this unit.
- Analysis of formal and informal student feedback from this unit.
- Comparison of attendance rates at computer laboratory classes before and after the introduction of this unit.
- Analysis of commentary relating to favourite assessment tasks in reflective reports.

4.1 Final Results and Attrition
The outcome of the assessment regime for the first offering of the unit was a unimodal distribution centred around 75%. Approximately 70% of students achieved a grade of 6 or 7, 20% achieved a grade of 4 or 5 and only 6% of the cohort failed to achieve a passing grade.

Updating Table 1 and Table 2 above gives some indication of the effect that pass rates may have had on attrition. Table 5 below shows that in semester 1, 2009 the failure rate dropped from 19% to 6%. Table 6 indicates that the attrition rate from the courses dropped

<table>
<thead>
<tr>
<th>Students</th>
<th>Semester 1</th>
<th>Semester 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Male</td>
<td>86%</td>
<td>80%</td>
</tr>
<tr>
<td>Female</td>
<td>14%</td>
<td>20%</td>
</tr>
<tr>
<td>Full Time</td>
<td>94%</td>
<td>93%</td>
</tr>
<tr>
<td>Part Time</td>
<td>6%</td>
<td>7%</td>
</tr>
<tr>
<td>Domestic</td>
<td>98%</td>
<td>99%</td>
</tr>
<tr>
<td>International</td>
<td>2%</td>
<td>1%</td>
</tr>
</tbody>
</table>

Table 4: Student Cohort Statistics

3.4.2 Learning Style
The VARK Learning Style questionnaire that students completed as part of their reflective report assessment provided a profile of their learning preferences in terms of giving and receiving information (Fleming 2009). Figure 1 below shows the breakdown of the learning styles of students in the first offering of INB104.

Figure 1: Student Cohort Learning Styles
from 35% to 9% while the attrition rate from the first programming unit has dropped from 19% to 6%.

<table>
<thead>
<tr>
<th>Result</th>
<th>First Programming Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sem 1, 2008</td>
</tr>
<tr>
<td>pass</td>
<td>81%</td>
</tr>
<tr>
<td>fail</td>
<td>19%</td>
</tr>
</tbody>
</table>

Table 5: Fail/Pass Rates

<table>
<thead>
<tr>
<th>Withdrawal</th>
<th>Attraction Rates</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sem 1, 2008</td>
</tr>
<tr>
<td>Changed to other course</td>
<td>16%</td>
</tr>
<tr>
<td>Discontinued enrolment</td>
<td>18%</td>
</tr>
<tr>
<td>Withdraw 1st Programming Unit</td>
<td>19%</td>
</tr>
</tbody>
</table>

Table 6: Attraction Rates

There has been a slight increase in the number of students who submitted assignments and who undertook weekly quizzes, indicating that the assessment tasks chosen for the unit seem to have been engaging to the student cohort.

4.2 Plagiarism

An unexpected but welcome benefit of the approach being used has been a reduction in plagiarism of assignment work. There have been only two instances of plagiarism so far in the new unit. This is a dramatic reduction when compared to previous offerings where there have typically been five to ten cases of plagiarism detected.

The reason for this has not been fully investigated but the most likely reason is the collaborative learning environment.

4.3 Student Feedback

To measure the success of the design of Building IT Systems we have collected formal and informal student feedback. Some student commentary is reproduced below. While there has been some feedback from students that has been negative, the majority of the comments have praised the unit. This is definitely the case with students who identified themselves through their comments as having failed the predecessor unit.

Students also provided positive feedback in their reflective reports about their learning experiences:

*I have definitely changed my mind slightly about several aspects after completing this course. I thought the programming part would be extremely hard and boring. I was very wrong about this and the programming parts didn’t turn out to be as difficult as I expected and they were more challenging and exciting than boring.*

Students were invited to provide feedback through the university’s Learning Experience Survey (LEX). The response rate from students in the unit was over 37%. Students were asked a number of questions using a five-point Likert scale, while also given the opportunity to provide free form qualitative comments:

*This unit wastes no time in getting right into the fun stuff, getting you engaged and making you participate. The problems were complex but solvable and the tutors and lecturers were always willing to help.*

The unit has been useful in providing me with skills to develop my programming expertise. The collaborative approach has been very useful for me personally. The whole unit is structured well. I personally found it engaging, as it had a great many ways to learn the topic. Going to lectures, listening to lectures via recording, the tutorial exercises and going to the practical for extra help, you can’t lose.

The PASS (Peer Assisted Study Scheme) leader for this unit also provided interesting feedback about the collaborations:

*I think the [pair] programming team idea is working very well, I have had a fair few couples come in and talk to me during pass sessions, and they often manage to figure out answers as a pair with a little push in the right direction. From what I have seen it is also stopping people being lazy...[I’ve seen some very motivated groups].*

4.4 Workshop Attendance

Historically in first year IT subjects at QUT, attendance at scheduled lectures, tutorials and practicals dramatically declines through the teaching semester. For example in 2007, an average of 80% of students initially attended programming practicals, and by the end of semester only 16% of students turned up. A similar pattern has been recorded for subsequent semesters (Teague and Roe 2009). Figure 2 displays a comparison of practical attendance rates from previous semesters for the first programming unit at QUT and for the first two offerings of INB104 in 2009.

![Figure 2: Practical Attendance Rates](image)

Attendances were recorded for 50% of the practicals in both semester 2, 2007 and semester 1, 2008. For the subsequent semesters in 2009, attendances were recorded for the entire cohort. A marked improvement in attendance rates by students in the new unit has been evidenced.

In semester 1, 2009, attendances were not collected in two of the practicals, one in week 10 and the other in week 12. For these two practicals, the attendance averages were 24 and 23 students respectively, each equating to approximately 7% of the entire cohort of enrolled students. Therefore, the actual attendance percentages for weeks 10 and 12 was more likely 7% higher than recorded in the graph. In week 4 of semester
2, 2009 the significant drop in attendances is due to a public holiday.

The reason for improved attendance at weekly computer laboratory practical sessions seems to be linked to the assessment tasks that have been used. This is discussed further in the next section regarding evidence collected from the reflective reports.

4.5 Analysis of Reflective Reports

The reflective reports were intended to highlight to the students the amount of learning that was achieved by participation in INB104. This was elicited from students through a series of questions.

The reflective reports have also provided insights from the students as to what engaged them in the unit. These insights indicate that the use of the portfolio of activities as the main assessment item for the unit and allowing students to choose which project tasks to include in their portfolios was a major reason for the success of the unit in reducing attrition and for increasing the pass rate.

Approximately 20% of students made some comment in their reflective reports about the specific projects that they undertook in the unit which they found most fun and engaging. Table 7 below shows that the projects with some visual feedback component were favoured by the students in the unit. We believe that this is not unexpected as the visual result indicates the successful completion of the project.

<table>
<thead>
<tr>
<th>Project Topic</th>
<th>Preferred Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>PyGame Animation</td>
<td>36%</td>
</tr>
<tr>
<td>Database Manipulation</td>
<td>10%</td>
</tr>
<tr>
<td>Turtle Graphics</td>
<td>41%</td>
</tr>
<tr>
<td>Popularity Cloud (HTML production by Python with data from a database)</td>
<td>33%</td>
</tr>
</tbody>
</table>

Table 7: Students’ Preferred Projects

Further analysis of the reflective reports shows that 92% of students made some comment in their reflective reports about the specific projects that they undertook in the unit which they found most fun and engaging.

### Future Plans

The next iteration of INB104 (this current teaching period) introduces an end of semester exam which will contribute 30% to a student’s final grade. The exam will attempt to test a working understanding of the basic concepts introduced during the semester covering programming, SQL, HTML and how these technologies inter-relate. Plenty of time will be provided to complete the exam, and enable them to perform to the best of their ability and to experience university exam conditions with low stress.

None of the introductory core units in the new course initially incorporated an exam as an assessment item. It was felt that INB104 was the logical unit among these core units for an exam to be used as a type of assessment that first year students should be exposed to. Examinations provide an alternative yet valid method for measuring learning outcomes in a compressed time frame with minimal opportunities for plagiarism.

Marking of the portfolio projects was very time-consuming, with both source code files and often quite verbose documentation to wade through and markers were instructed to provide significant and effective feedback to the students. The current iteration of the unit will have only one draft portfolio before the final version is submitted. This will reduce the marking workload on teaching staff, but still provide students with valuable feedback in order to both improve their submission, and prepare for the final exam.

Students loved the “fun stuff”, and that meant something different for each student. The Games degree students tended to prefer Turtle graphics and the task using the PyGame library, while others found joy in building a database or creating Web pages using Python. While the projects will generally be recycled in subsequent semesters, new projects will also be developed, providing a wider range of project options for students. Ideas for new projects include making use of readily available libraries for Python.

Students were generally very interested in their learning style profiles provided by the VARK Learning Style questionnaire. In the current and future offerings of this unit, students will be required to complete the learning style questionnaire in week 1, which gives them the opportunity to immediately take advantage of the information they gain about their learning preferences during their study.

### Conclusions

In response to falling student numbers in the Bachelor of IT degree at QUT, and high failure rates and high attrition rates in introductory computer programming units, the faculty undertook a major course revision which included significant changes to the first programming unit encountered by first year students.

We believe that the changes made to the unit have resulted in better engagement with the material by the students. Attrition from the unit has been reduced to 6%. The failure rate in the unit has also been reduced to 6%.

This has been achieved while maintaining introductory computer programming concepts – statement sequences, conditional statements, iterative and recursive approaches.
to repetition, and functional decomposition using top-down design.

The unit has also introduced some of the basic concepts of database design and manipulation, networking, and Web page production using HTML. We believe that we have covered the core fundamentals of these topics in this unit, giving students sufficient grounding in these areas so they are aware of them in their professional lives, even if they will not directly use those skills.

Workshops were conducted in a pair programming mode so that students could learn in a collaborative manner, being able to support each other’s learning. Informal and formal feedback indicates that this has been well received by the students.

Data collection from student assessment supports our claim that project tasks have been designed to be engaging. We believe that this is because many of the projects have a visual output, and that they are extensible so that students with prior knowledge in the area can demonstrate their higher level skills.

For assessment, students have undertaken weekly quizzes and produced a portfolio of the collaborative project tasks undertaken and have written a reflective report, outlining the skills and knowledge gained during the semester. The reflective report has also required students to articulate their preferences with regard to the topics covered in the unit. This has afforded them the opportunity to contemplate the different study paths that are offered from which they can select topics to study for the remainder of their course. We believe that this may also lead to lower attrition rates and lower failure rates for follow on units, as students have written in their reflections that they will not enrol in units that they now know to be personally unappealing.

7 References


