Pedagogy of Introductory Computer Programming: 
A People-First Approach

by

Donna Teague (BIT)

A thesis submitted in partial fulfilment 
of the requirements for the degree of

Master of Information Technology (Research)

at the

Queensland University of Technology

2011
Keywords

Introductory Programming
Learning to Program
Programming Pedagogy
Collaborative Learning
Pair-Programming
Abstract

Students struggle with learning to program. In recent years, not only has there been a dramatic drop in the number of students enrolling in IT and Computer Science courses, but attrition from these courses continues to be significant. Introductory programming subjects traditionally have high failure rates and as they tend to be core to IT and Computer Science courses can be a road block for many students to their university studies.

Is programming really that difficult — or are there other barriers to learning that have a serious and detrimental effect on student progression?

In-class experiments were conducted in introductory programming units to confirm our hypothesis that that pair-programming would benefit students' learning to program. We investigated the social and cultural barriers to learning programming by questioning students' perceptions of confidence, difficulty and enjoyment of programming. The results of paired and non-paired students were compared to determine the effect of pair-programming on learning outcomes.

Both the empirical and anecdotal results of our experiments strongly supported our hypothesis.
Contents

Chapter 1. Introduction ........................................................................................................ 1
  1.1 Motivation .................................................................................................................. 1
    1.1.1 Background ...................................................................................................... 2
    1.1.2 Student Behaviour ......................................................................................... 4
  1.2 Research Question .................................................................................................... 7
  1.3 Research Objectives .................................................................................................. 7
  1.4 Hypothesis ................................................................................................................ 8
  1.5 Overview .................................................................................................................. 8
    1.5.1 Literature Review ............................................................................................ 9
    1.5.2 Method ............................................................................................................. 9
    1.5.3 Result Summary .............................................................................................. 10
    1.5.4 Outcomes and Contributions ......................................................................... 11
  1.6 Conclusion ................................................................................................................. 11

Chapter 2. Literature Review ............................................................................................ 12
  2.1 Constructivism .......................................................................................................... 12
  2.2 Documented Pedagogy to Improve Learning Outcome ............................................ 13
    2.2.1 Encouraging Hands-On Practice and Experimentation ................................... 14
    2.2.2 Providing Multiple Entry Paths into Courses .................................................. 14
    2.2.3 Addressing Student Motivation and Engagement .......................................... 15
    2.2.4 Addressing Confidence, Enjoyment and Interest .......................................... 17
    2.2.5 Using Tools and Tutorial Systems .................................................................. 19
  2.3 Social and Cultural Barriers to Learning ................................................................. 20
    2.3.1 External Commitments .................................................................................... 21
    2.3.2 Cultural Perceptions ....................................................................................... 22
    2.3.3 Generation Y Culture ...................................................................................... 23
Chapter 3. Methodology

3.1 Introduction

3.2 Multi Method

3.3 Data Collection

3.3.1 Quantitative Data: In-Class Experimental Studies

3.3.1.1. ITB001 — Problem Solving & Programming (Semester 2, 2007)

3.3.1.2. ITB003 — Object Oriented Programming (Semester 2, 2007)

3.3.1.3. ITB001 — Problem Solving & Programming (Semester 1, 2008)

3.3.1.4. INB104 — Building IT Systems (Semester 1, 2009)

3.3.1.5. INB104 — Building IT Systems (Semester 2, 2009)

3.3.1.6. INB104 — Building IT Systems (Semester 1, 2010)

3.3.1.7. Summary of Teaching Units

3.3.1.8. Workshop Pairing

3.3.2 Qualitative Data

3.3.2.1. Then & Now Survey

3.3.2.2. Case Studies
3.3.2.3. Weekly Surveys ...........................................................................70

3.4 Summary ...............................................................................................71

Chapter 4. Results ...................................................................................... 72

4.1 Introduction .............................................................................................72

4.2 Social and Cultural Experiences .............................................................72
   4.2.1 Then & Now Survey .........................................................................72
   4.2.2 Case Studies .....................................................................................75
      4.2.2.1. Nelly [studying ITB001] .............................................................76
      4.2.2.2. Jane [studying ITB001] ...............................................................78
      4.2.2.3. Dave [studying ITB003] .............................................................82
      4.2.2.4. Steve [studying ITB001] .............................................................85
      4.2.2.5. Summary of Case Studies .........................................................88
   4.2.3 Weekly Surveys ................................................................................90

4.3 Pair-Programming Experiments .............................................................92
   4.3.1 Quantitative Results .........................................................................93
      4.3.1.1. Participation in Initial Experiments .............................................93
      4.3.1.2. Participation in the Replicated Experiments ...............................94
      4.3.1.3. Student Results .......................................................................95
      4.3.1.4. Threats to Quantitative Analysis .................................................98
      4.3.1.5. An Indicator of Expected ITB001 Results .................................99
   4.3.2 Qualitative Results ..........................................................................101
   4.3.3 Improvements to Pair Selection .......................................................108
   4.3.4 Improvements to Pair-Programming Protocol .................................110

4.4 Summary ...............................................................................................113

Chapter 5. Conclusions and Future Work ....................................................115

5.1 Outcomes ..............................................................................................116

5.2 Future Work .........................................................................................117

Appendices .................................................................................................119
<table>
<thead>
<tr>
<th>Appendix</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>ITB001 Unit Outline — Semester 2, 2007</td>
<td>119</td>
</tr>
<tr>
<td>B</td>
<td>ITB003 Unit Outline — Semester 2, 2007</td>
<td>120</td>
</tr>
<tr>
<td>C</td>
<td>ITB001 Unit Outline — Semester 1, 2008</td>
<td>121</td>
</tr>
<tr>
<td>D</td>
<td>INB104 Unit Outline — Semesters 1, 2009 – 1, 2010</td>
<td>122</td>
</tr>
<tr>
<td>E</td>
<td>Collaborative Learning Information Sheet</td>
<td>123</td>
</tr>
<tr>
<td>F</td>
<td>Then &amp; Now Perception Survey (Semester 1, 2007)</td>
<td>125</td>
</tr>
<tr>
<td>G</td>
<td>Interviewee Profile Form</td>
<td>126</td>
</tr>
<tr>
<td>H</td>
<td>Interview Form</td>
<td>127</td>
</tr>
<tr>
<td>I</td>
<td>Weekly Survey (Semester 2, 2007)</td>
<td>128</td>
</tr>
<tr>
<td>J</td>
<td>Published Papers</td>
<td>129</td>
</tr>
</tbody>
</table>

References | 130
List of Figures

Figure 1.1: Programming Failure Rates from 2003 .................................................. 3
Figure 1.2: ITB001 Workshop Attendance Rates Semester 1, 2007 .......................... 4
Figure 1.3: ITB001 Workshop Attendance Rates Semester 2, 2007 .......................... 5
Figure 1.4: ITB001 Workshop Attendance Rates Semester 1, 2009 .......................... 5
Figure 2.1: Employment & Study Commitments ..................................................... 21
Figure 2.2: Workforce and Education ................................................................. 22
Figure 3.1: Data Collection Periods .................................................................. 59
Figure 4.1: Change in Student Perception of Unit Difficulty .................................. 73
Figure 4.2: Change in Student Perception of Enjoyment of Unit .............................. 73
Figure 4.3: Change in Student Confidence ......................................................... 74
Figure 4.4: Student Perception of Collaborative Learning ..................................... 75
Figure 4.5: Confidence and Determination .......................................................... 89
Figure 4.6: Student Weekly Perceptions ............................................................... 90
Figure 4.7: Correlation Coefficients of Confidence, Enjoyment & Difficulty .......... 91
Figure 4.8: Student Study Behaviours ................................................................. 92
Figure 4.9: Failure Rates During Experiments ...................................................... 95
Figure 4.10: Distribution of Grades During Initial Experiment ............................... 96
Figure 4.11: Failure Rates of Paired and Unpaired Students .................................. 96
Figure 4.12: Analysis of Variance Results ........................................................... 97
Figure 4.13: Exam Performance ........................................................................... 98
Figure 4.14: Difference in Grades for Paired and Unpaired Students ..................... 100
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1.1</td>
<td>Glossary of Terms</td>
<td>x</td>
</tr>
<tr>
<td>Table 1.2</td>
<td>Acknowledgements</td>
<td>xii</td>
</tr>
<tr>
<td>Table 2.1</td>
<td>Summary of Empirical Pair-Programming Research</td>
<td>52</td>
</tr>
<tr>
<td>Table 2.2</td>
<td>Issues Addressed with Collaborative Learning</td>
<td>54</td>
</tr>
<tr>
<td>Table 3.1</td>
<td>Establishing Logistics of Pair-Programming based on Key Findings</td>
<td>57</td>
</tr>
<tr>
<td>Table 3.2</td>
<td>Comparison of Teaching Units</td>
<td>64</td>
</tr>
<tr>
<td>Table 3.3</td>
<td>Pair-programming Protocols in Classroom</td>
<td>65</td>
</tr>
<tr>
<td>Table 4.1</td>
<td>Paired t-Tests on Change in Perceptions</td>
<td>74</td>
</tr>
<tr>
<td>Table 4.2</td>
<td>Case Study — Nelly: Pre-Unit Perceptions</td>
<td>76</td>
</tr>
<tr>
<td>Table 4.3</td>
<td>Case Study — Nelly: Study versus Work</td>
<td>76</td>
</tr>
<tr>
<td>Table 4.4</td>
<td>Case Study — Nelly: Post-Unit Perceptions</td>
<td>78</td>
</tr>
<tr>
<td>Table 4.5</td>
<td>Case Study — Nelly: Grade Expectation versus Actual</td>
<td>78</td>
</tr>
<tr>
<td>Table 4.6</td>
<td>Case Study — Jane: Pre-Unit Perceptions</td>
<td>79</td>
</tr>
<tr>
<td>Table 4.7</td>
<td>Case Study — Jane: Study versus Work</td>
<td>79</td>
</tr>
<tr>
<td>Table 4.8</td>
<td>Case Study — Jane: Post-Unit Perceptions</td>
<td>81</td>
</tr>
<tr>
<td>Table 4.9</td>
<td>Case Study — Jane: Grade Expectation versus Actual</td>
<td>82</td>
</tr>
<tr>
<td>Table 4.10</td>
<td>Case Study — Dave: Pre-Unit Perceptions</td>
<td>82</td>
</tr>
<tr>
<td>Table 4.11</td>
<td>Case Study — Dave: Study versus Work</td>
<td>83</td>
</tr>
<tr>
<td>Table 4.12</td>
<td>Case Study — Dave: Post-Unit Perceptions</td>
<td>84</td>
</tr>
<tr>
<td>Table 4.13</td>
<td>Case Study — Dave: Grade Expectation versus Actual</td>
<td>84</td>
</tr>
<tr>
<td>Table 4.14</td>
<td>Case Study — Steve: Study versus Work</td>
<td>86</td>
</tr>
<tr>
<td>Table 4.15</td>
<td>Summary of Workshops and Pairings in Experiments</td>
<td>93</td>
</tr>
<tr>
<td>Table 4.16</td>
<td>QUT Grade Cut-Offs</td>
<td>100</td>
</tr>
</tbody>
</table>
## Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Explanation / Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUT</td>
<td>Queensland University of Technology, Brisbane, Queensland, Australia</td>
</tr>
<tr>
<td>IT</td>
<td>Information Technology</td>
</tr>
<tr>
<td>CS</td>
<td>Computer Science</td>
</tr>
<tr>
<td>Unit</td>
<td>A course of study (a subject), normally conducted over one semester.</td>
</tr>
<tr>
<td>Semester</td>
<td>A teaching period of approx 13 weeks. QUT runs 3 semesters per year: Semester 1 = February to May Semester 2 = July to October Summer Semester = November to February</td>
</tr>
<tr>
<td>Blackboard</td>
<td>Web-based Learning Management System which gives staff and students access to online learning materials for each of their units at QUT</td>
</tr>
<tr>
<td>ITB001</td>
<td>QUT teaching unit. Problem Solving and Programming. First introductory programming and core unit offered in the Bachelor of IT degree course.</td>
</tr>
<tr>
<td>ITB003</td>
<td>QUT teaching unit. Object Oriented Programming. Second introductory programming and core unit offered in the Bachelor of IT degree course.</td>
</tr>
<tr>
<td>INB104</td>
<td>QUT teaching unit. Building IT Systems. New core unit in Semester 1, 2009 touching on a range of technologies including programming, databases and web development.</td>
</tr>
<tr>
<td>Scheme</td>
<td>Programming language derived from Lisp. Best known for its support of the functional paradigm. This language was used in ITB001 in 2007 with the IDE, DrScheme. <a href="http://www.drscheme.org/">http://www.drscheme.org/</a></td>
</tr>
<tr>
<td>Python</td>
<td>Object oriented programming and scripting language which also supports the functional paradigm. This language was used in ITB001 in 2008 and INB104 in 2009 with the IDE, IDLE. <a href="http://www.python.org/">http://www.python.org/</a></td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>Pair learning</td>
<td>When another’s point of view or approach to solving a problem may not have occurred to the other; when skill levels are uneven and there is transfer from one to the other</td>
</tr>
<tr>
<td>Pair pressure</td>
<td>Peer pressure and commitment within a pair to stay on task and perform</td>
</tr>
<tr>
<td>Pair courage</td>
<td>When individuals may not be confident enough to tackle a complex task, but with the support of another gives more confidence</td>
</tr>
<tr>
<td>Workshop / Practical / Tutorial</td>
<td>Interchangeable terms. Refers to a scheduled weekly session facilitated by teaching staff, normally for two hours which follows and supplements the weekly lecture for a unit. These sessions would normally be used to reinforce and experiment with the concepts introduced in the lecture.</td>
</tr>
<tr>
<td>Flat room</td>
<td>Teaching area devoid of (supplied) computer equipment.</td>
</tr>
<tr>
<td>Laboratory</td>
<td>Teaching area fitted with computer equipment.</td>
</tr>
<tr>
<td>I/O</td>
<td>Input/Output</td>
</tr>
<tr>
<td>OO</td>
<td>Object oriented</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical user interface</td>
</tr>
</tbody>
</table>

*Table 1.1: Glossary of Terms*
Statement of Original Authorship

The work contained in this thesis has not been previously submitted to meet requirements for an award at this or any other higher education institution. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made.

Submission of Final Thesis:

Signature:..............................................................................................................

Date:......................................................................................................................
Acknowledgements

I would like to thank a number of people who supported and encouraged me during this project including but not limited to my official supervisory team. Firstly thanks to Dr Peter Bancroft who convinced me I was good enough to embark on a research degree in the first place, and my principal supervisor Dr Paul Roe who remained confident in my ability and supportive of my endeavours over the years it took to complete the project. I would also like to thank Dr Glenn Smith who as associate supervisor for a period of the research also provided direction and a great deal of encouragement. My gratitude is also conveyed to Dr Raymond Lister and Malcolm Corney who each proof-read and critiqued my documentation a number of times, and helped with the drive to continue, and the wisdom to know when to stop.

I would like to acknowledge the efforts of the unit co-ordinators who allowed me to conduct this research, and their willingness to have me use pair-programming experiments in their workshops with their students. I need also acknowledge the involvement of other teaching staff who were involved in the pairing experiment at various times and collected pairing information during workshops they conducted.

<table>
<thead>
<tr>
<th>Year</th>
<th>Sem</th>
<th>Unit</th>
<th>Co-Ordinator</th>
<th>Other teaching staff involved in pairing experiment and data collection</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007</td>
<td>2</td>
<td>ITB003</td>
<td>Diane Corney</td>
<td>(Research student only)</td>
</tr>
<tr>
<td>2007</td>
<td>2</td>
<td>ITB001</td>
<td>Glenn Smith</td>
<td>(Research student only)</td>
</tr>
<tr>
<td>2008</td>
<td>1</td>
<td>ITB001</td>
<td>Colin Fidge</td>
<td>(Research student only)</td>
</tr>
<tr>
<td>2009</td>
<td>1</td>
<td>INB104</td>
<td>Malcolm Corney</td>
<td>Malcolm Corney, Ross Hayward, Jason Wimmer, Thet Ko</td>
</tr>
<tr>
<td>2009</td>
<td>2</td>
<td>INB104</td>
<td>Malcolm Corney</td>
<td>Malcolm Corney, Ross Hayward, Jason Wimmer, Thet Ko</td>
</tr>
<tr>
<td>2010</td>
<td>1</td>
<td>INB104</td>
<td>Malcolm Corney</td>
<td>Malcolm Corney, Ross Hayward, Matthew Brecknell, Jason Thorne</td>
</tr>
</tbody>
</table>

Table 1.2: Acknowledgements
Chapter 1. Introduction

Students struggle with learning to program. In recent years, not only has there been a dramatic drop in the number of students enrolling in IT and Computer Science courses (DEST 2006; Reges 2006; Lang, McKay et al. 2007; DEST 2008; Lister 2008; Slonim, Scully et al. 2008; Australian Bureau of Statistics 2009), but attrition from these courses continues to be significant (Beaubouef and Mason 2005; Biggers, Brauer et al. 2008). Introductory programming subjects traditionally have high failure rates and as they tend to be core to IT and Computer Science courses can be a road block for many students to their university studies.

Is programming really that difficult — or are there other barriers to learning that have a serious and detrimental effect on student progression?

1.1 Motivation

The dropout rate of from Queensland University of Technology (QUT)’s Bachelor of Information Technology degree in 2007 was 22%, with attrition rates from all courses at the same university as well as nationally at 17% for the same year (DEEWR 2009). Other universities report similarly high attrition rates from their CS programs, some even as high as 50% (Beaubouef and Mason 2005; Lang, McKay et al. 2007; Soh Leen-Kiat 2007).

The motivation for this research was as a result of my own recent completion of a Bachelor of Information Technology (BIT), Software Engineering (SOF) major. My personal experiences as a mature-age female student in such a young-male dominated course gave me insight into the social and cultural barriers that can prevent students from learning to program and piqued my interest in investigating possible remedies.

According to QUT corporate reports (2008), in 2005:

- females accounted for 14% of all BIT students (58:414);
- females accounted for 10% SOF students (28:272);
- female SOF students accounted for less than 7% of all IT students (28:414);

Not only were females poorly represented in the BIT, but the attrition rate was higher in this minority group, with 37% of females as opposed to 19% of male students dropping out (QUT 2010(2)). This reflected the trend in the US with reports of between 5% more and
twice as many women than men leaving IT at university (Cohoon 1999; Fisher and Margolis 2002).

As a female student in a male-dominated course, it was intimidating to hear other Software Engineering students talk about their programming experience when I had none. However, good time-management and study skills and an awareness of what was expected for each unit levelled the playing field. It was not until sometime through the course that I learned that their stories of vast previous experience had mostly been exaggerated, and that I had perhaps worried needlessly. However, many of the BIT students, especially the women, had no desire to be seen as fitting the geeky programmer stereotype and tended to lose interest in software engineering subjects and choose other majors.

During the early years of my study, I became involved in the university's Peer Assisted Study Scheme (PASS) which is based on the International Supplemental Instruction Model developed by the University of Missouri – Kansas City (2003). As a PASS leader, I provided peer support in subjects I had thus far completed. In the latter years of my degree, I worked in teaching and research assistant roles and eventually as a tutor for several undergraduate and postgraduate introductory programming units. I suspected that some of the issues that students encountered when learning to program were:

a) difficult content;
b) lack of enjoyment;
c) inability to see relevance, or "real world" application of the unit content;
d) lack of motivation to learn;
e) lack of confidence in ability to learn;
f) reluctance to approach academic staff for help, especially outside workshop times;
g) reluctance to talk to peers for fear of breaching plagiarism rules; and
h) "getting stuck" on programming concepts and being unable to resolve them without help.

1.1.1 Background
Programming is acknowledged by tertiary educators to be an inherently complex intellectual activity, with students struggling through their first programming subject and educators struggling to teach it (McCracken, Almstrum et al. 2001; Bruce and McMahon 2002; Robins, Rountree et al. 2003; Lister, Adams et al. 2004; Lahtinen, Ala-Mutka et al. 2005). The high failure rate of first year programming students has for many years been a controversial topic for learning institutions with reports of failure rates in the vicinity of
26% to 40% (Sheard and Hagan 1998; Truong, Bancroft et al. 2003; Lang, McKay et al. 2007; Han and Beheshti 2010) and one for which further insight into the causes and/or contributing factors would be valuable to both universities and the students themselves.

Since 2003, QUT has experienced a high failure rate of students in its introductory programming unit: averaging 30.9% and peaking at 41% in late 2006. (See Figure 1.1 below which shows the failure rates for all first semester units at QUT since 2003, with IT8001/111 being the programming unit).

At QUT, the first programming unit has consistently had the worst failure rate of any first semester, first year IT unit (Corney, Teague et al. 2010). This is by no means peculiar to QUT with high failure rates also reported at other universities (Woszczynski, Haddad et al. 2005; Bennedsen and Caspersen 2007; Lang, McKay et al. 2007; Gomes and Mendes 2010).

Much debate has ensued regarding the approaches, content and structure of computer science (CS) subjects (Gonzalez 2006; Schulte and Bennedsen 2006) including choice of:

a) programming language (e.g., C, C++, Java, C#, Scheme);

b) programming paradigm (e.g., imperative, object-oriented, functional);

c) Integrated Development Environment (IDE) (e.g., VisualStudio, Eclipse, BlueJ);

d) course complexity and breadth; and

e) topics to be covered at the introductory level and the order in which those topics are introduced.
In an effort to remedy the continuing high failure and attrition rates, universities have tended to adjust one or more of the above choices. QUT has in the last decade for example tried a variety of programming languages in which to teach introductory programming units, from Modula2, Java, C# and Scheme to Python. It seems that most efforts have been directed according to an apparent perception that programming is just too difficult for students to master, and that the main reason for failure lies in the inability of those students to ever understand the unit content.

With high failure rates in introductory programming units, high drop-out rates and shrinking enrolments in Computer Science courses, the primary motivation for investigating the pedagogy of learning computer programming was to better understand what could be done to improve the learning outcomes for more programming students.

1.1.2 Student Behaviour

Consistently in first year IT subjects at QUT, attendance levels at scheduled lectures and workshops\(^1\) dramatically decline through the semester. In the first week of Semester 1, 2007, the introductory programming subject, ITB001, saw on average 80% of students attending workshops, and by the end of semester the average attendance rate at workshops was only 16% (see Figure 1.2 below). (Note that the Friday of Week 6 was a public holiday.)

![Attendance rates graph](image)

Figure 1.2: ITB001 Workshop Attendance Rates Semester 1, 2007

---

\(^1\) Reference to "practical", "workshop" or "tutorial" should be read as interchangeable. They each refer to a weekly class facilitated by teaching staff either in a flat room or computer laboratory.
Subsequent semesters experienced a similar pattern of attendance as highlighted in Figure 1.3 and Figure 1.4 below.

(Note that the Monday of Week 10 in Semester 1, 2008 was a public holiday.)

Introductory programming students at QUT have been reluctant to seriously embrace time-management advice offered by academic staff for successfully completing the subject. A "devil may care" attitude sees students deferring any significant effort or focus on the course material until late in semester. Not unexpectedly, many of them end up struggling to complete complex programming projects in a very limited amount of time. They find themselves with little of the working knowledge required to solve the assessment task.
Elevated stress levels compound the problem often resulting in the student’s inability to successfully complete the assessment item in time.

Poor grounding in the "building block" basics of programming like variable declaration, function definition and parameter passing in the early weeks of semester make the more advanced topics of loops, recursion and abstract data types almost impossible to grasp. Even with the "wake-up" call of a failed first assessment item and a renewed enthusiasm for putting in some real effort, there is all too often little chance to catch up on the workload in time to salvage a decent grade for the subject. The student is in danger of losing confidence in their own ability and disengaging from the subject altogether.

Why do students fail to engage in the first place? Take a typical Australian first year IT student: an 18 year old male, techno-savvy school leaver, fresh out of a three month summer vacation; having spent the last 12 years learning experience closely supervised by parents and teachers. The contrast is stark between high school as a "child" and attending the adult world of university. Students find themselves with the sole responsibility for their learning and this presents the often immature adolescent with the power to absent themselves from university commitments and succumb to the allure of a heightened social existence (Begley 2000).

But it is not only school-leavers who fail to engage. Those students who don’t fit the IT student stereotype include not only women, but mature-age students and others who see studies in IT as complementary to their career aspirations, rather than the focus thereof (Vilner and Zur 2006; Peckham, Stephenson et al. 2007). These students may initially have a better study ethic, but can struggle with a lack of supporting social structure in the learning environment (Cohoon 2002) and disinterest in or inability to relate to the learning material (Fisher and Margolis 2002). It may not necessarily be a problem with what is taught in introductory programming units, but how it is presented (Gonzalez 2006).

The sequential nature of learning to program may mean that a concerted and consistent effort is needed by students over the whole semester, and this unforgiving nature of the subject is often what challenges students.

A recent study of CS1 minor students (Kinnunen and Malmi 2006) found a large variety of reasons for students dropping out of a CS1 course, with the major reasons being:

- perceived difficulty of course;
• lack of motivation:
  • to study in general;
  • because payoff seems imbalanced; or
  • because course seems too difficult;

• lack of time:
  • or at least time management and planning skills;
  • as course is too difficult, not enough time to complete it successfully; or
  • not committed to time required in the first place.

Treisman (in Seymour and Hewitt 1997) reports from a study of calculus students that the key elements of success in science, mathematics and engineering (SME) were:

• students’ awareness of their teacher’s high expectations;
• group study and support;
• shared experience of success in problem solving of progressively more difficult nature; and
• building of self-confidence.

Seymour & Hewitt (2004) conclude that:

"... SME attribution cannot be viewed as a natural consequence of differential levels of ability; classroom climate and activities play critical roles in determining the students who do, and do not, persist within SME majors."

The issues of student confidence, enjoyment and motivation, and collaborative interaction with peers have been identified as contributing to student success in learning to program, and these key areas will be the focus of this research.

1.2 Research Question

This research investigates the contributing factors to high failure rates and high attrition by studying introductory programming students and experimenting with the way they engage and interact in their learning environment. The research question is:

*Does pair-programming as a pedagogical approach for learning to program improve learning outcomes?*

1.3 Research Objectives

The main objectives identified from the research question are to:
a) Learn more about student cohorts and their perceptions, attitudes and motivations for studying programming;

b) Develop a collaborative learning environment for programming which improves confidence and motivation to deal with difficult concepts, and increases the enjoyment of learning to program. We aim to continuously improve the environment through several cycles of pair-programming in the classroom, refining pair-programming protocols and the process of pair-selection to maximize the benefit to students.

c) Determine if improving confidence, motivation and enjoyment of learning to program improves students' learning outcomes.

1.4 Hypothesis

We hypothesise that pair-programming has a positive effect on learning outcomes by addressing social and cultural issues that students face while learning to program.

1.5 Overview

Students' practical experience of programming at university is often restricted to individual work during class and individual assessment items. The culture of learning to program is one of competition and individualism (Werner, Hanks et al. 2004; Bagley and Chou 2007), rather than collaboration and co-operative work. "Group work" is often used to describe the process of completing a project or large task by assigning smaller tasks to members of the group to complete individually, and piecing together those individual tasks when completed. Members often choose tasks which they feel they have more competence or interest in, and it is not unusual to see more experienced and confident members dominating the project.

Students generally dislike group assignments, as they contain some element of reliance on potentially unreliable team mates, and have the tendency to promote parasitic behaviour by weaker, less motivated students in the group. For these reasons also, group work assessment has the reputation of being an unreliable indication of each member's level of achievement and is rarely included in introductory programming units.

Another form of collaboration, "pair-programming", which will be described in more detail in later chapters, has both members of a team completing every aspect of the project, one task at a time, using shared resources. There is little or no division of work as members work as one. Pair-programming was used in this research to address the issues of
confidence, difficulty of learning topic and enjoyment of programming with the measures of success being improved student learning outcomes and reduced failure rates.

The following sub-sections give brief summaries of the remaining chapters of this Thesis.

1.5.1 Literature Review

Chapter 2 starts with a brief background discussion on learning theory, the issues of learning to program and strategies used to address these issues to date. The chapter then turns to “people focused” approaches to programming pedagogy which concentrates on the students themselves (the people) by looking at collaboration and discussing the known benefits of using this type of learning environment. From there, the focus is on collaborative programming which leads to the more specific pair-programming and its application in industry as well as learning environments.

1.5.2 Method

This section provides a brief description of the methodology used in this research, full details of which are included in Chapter 3.

This research started its lifecycle with a survey of introductory programming students at QUT in Semester 1, 2007 in an effort to discover the “people issues” behind learning to program. Students were queried about their perceptions in terms of:

1. how confident they were in successfully learning to program;
2. how difficult they were finding programming; and
3. how much they were enjoying programming.

The survey took place during mid-semester and required students attending the weekly lecture to provide their current perceptions on these three key areas, as well as recall their perceptions at the beginning of semester in Week 1. The students were also questioned about their perceptions of collaborative learning.

After analysis of the students’ feedback from those surveys, collaborative learning experiments were implemented using pair-programming in two of QUT’s first year programming units. Details of the teaching units can be found in Section 3.3.1. The author was a member of the teaching staff of both of these units.

Experiments were conducted over several semesters spanning 2007 and 2008, and involved participation by students who had allocated themselves to workshops facilitated by the author. The student subjects exercised the pair-programming protocols discussed at length
in the literature review in Chapter 2. Although the primary teaching unit used for this research was dramatically redeveloped for the start of the 2009 teaching year, data continued to be collected and observations made during replicated experiments spanning 2009 and 2010.

The initial pair-programming experiments were complemented with case studies that gathered more sociological data about a number of students from the cohort targeted in this research. The entire cohort of students were surveyed on a weekly basis to monitor their perceptions of programming confidence, difficulty and enjoyment as they progressed through the semester. In subsequent semesters of the experiment, other qualitative information was gathered from students via end of semester reflective reports.

1.5.3 Result Summary

The initial retrospective survey of student perceptions showed that overall students were finding programming more difficult than they had anticipated; they were enjoying it less than they had hoped, and were less confident about being able to learn to program. In terms of students’ perceptions of collaborative learning, the majority of them believed that such an environment would not only have a beneficial impact on their learning outcomes, but also make studying programming more engaging, interactive, and fun.

In the following semester, the weekly surveys that the entire student cohort completed showed a relationship between the difficulty of programming and both confidence and enjoyment levels. When the work was more challenging, they were less confident about being able to successfully learn to program, and they enjoyed the learning process less.

The case studies of introductory programming students revealed that the common issues most likely affecting student outcomes were confidence and motivation, with level of ability being less relevant to successfully learning to program.

During the pair-programming experiments, most students embraced the collaborative learning environment for their programming studies, which helped address issues of confidence (over and under), motivation, engagement and enjoyment. The constant peer interaction also encouraged students to refine their time management, study and communication skills.
The encouraging qualitative and quantitative results strongly suggest that collaborative learning by pair-programming addressed "people issues" and had a positive effect on the students' learning outcomes. Detailed results and analysis can be found at Chapter 4.

1.5.4 Outcomes and Contributions
This research showed us that the barriers students face while learning to program include lack of confidence in programming ability and lack of enjoyment of the learning process when the content was difficult.

We have confirmed the benefits of collaborative learning and more particularly, pair-programming with far fewer students failing introductory programming when they learned in this type of environment. Students reflected that while using pair-programming to learn, they were more confident in their programming ability, enjoyed learning to program and were able to complete more difficult programming tasks than they might otherwise have done alone.

Repeated experiments over several teaching periods used pair-programming in the classroom and iteratively refined the pair selection process as well as the protocols used in order to improve students' confidence and enjoyment of learning to program, and to maximise their learning outcomes. Full details of the outcomes of this research and conclusions drawn can be found at Chapter 5.

1.6 Conclusion
This chapter has described the issues of learning to program and the continuing high failure rates in introductory programming units and high attrition rates from IT and Computer Science courses. The motivation for investigating the social and cultural issues presenting to programming students was covered, together with an overview of the work involved in this research.

The next chapter provides a detailed literature review of the works relating to the pedagogy of learning to program.
Chapter 2. Literature Review

Chapter 1 has highlighted that learning to program is difficult and students struggle to pass introductory programming units. With programming as a core unit in computer science courses, high failure rates compound the issue of attrition. Less interest by school leavers in computer science evidenced by weaker enrolment figures suggest that there may be social and cultural barriers to learning programming that need to be addressed.

The main objectives of this research were to learn about student perceptions of programming, and to develop and measure the effect of a collaborative learning environment which addresses the negative perceptions that may otherwise have been detrimental to learning outcomes.

This chapter introduces the main learning theory used in programming pedagogy in Section 2.1 along with various documented approaches others have used to improve the progression of students in computer science courses in Section 2.2. Teaching computer programming is more than simply having students learn syntax and structure and this research aims to address people issues. Section 2.3 discusses social and cultural barriers to learning and learning computer programming in particular. Collaborative learning has been identified as the major thrust of this research and Section 2.4 discusses collaborative learning, its benefits and collaborative approaches to programming particularly in a learning environment.

2.1 Constructivism

Constructivism is a theory based on the premise that a student actively engages in a learning activity by integrating new information, and building knowledge and skills based on prior knowledge and experience rather than just passively absorbing what is presented to them. Piaget (1952) and Vygotsky (1986) are credited with establishing this theory. Piaget's cognitive constructivism focuses on intellectual growth through age related development while Vygotsky's perspective on constructivism is more concerned about cultural and social contexts playing a fundamental role in cognitive development.

Bruner (1990) and Huitt (2003) did much to disseminate the ideas of constructivism, which seems to be the predominant teaching and learning paradigm in education today.
The philosophical and epistemological assumptions held by constructivists have been summarised by Muise and Wakkary (2010) as follows:

- A real world exists that acts as a boundary for what an individual can experience. Despite this, reality exists in the mind of the individual, necessitating multiple realities – one for each individual.
- The structure of reality is created in the mind through interacting with the world. The structuring of reality occurs through the use of symbols.
- The mind creates symbols by perceiving and interpreting the world.
- Human thought is developed through perception, sensory experiences, and social interaction.
- Meaning occurs through an interpretive process that is dependent upon an individual’s previous experiences and understandings.

According to this theory, the process of learning for each student is reliant on pre-existing knowledge and experience and the integration of new material and context. Knowledge does not exist independent of the learner (Vrasidas 2000), but is constructed during this process.

According to Hsiao (2007), the goal of constructivism is to create learning communities of teachers and students more closely related to the collaborative practice of the real world where students assume responsibility for their own learning. Collaboration involves the participants viewing problems from different perspectives and being able to negotiate and generate meanings and solutions through shared understanding.

2.2 Documented Pedagogy to Improve Learning Outcome

With introductory programming a foundation unit for most IT/CS courses, many of the students are still at a stage of transitioning into university. The first year at university is a crucial time for engaging students and equipping them with the skills to be successful and an Australian Learning and Teaching Council fellowship project (Kift 2009) provides some guidance for addressing transitioning issues at university:

"Learning, teaching, and assessment approaches in the first year curriculum should enact an engaging and involving curriculum pedagogy and should enable active and collaborative learning"
The sections that follow document some of the approaches taken to improve the learning outcome for first year students and to address the high attrition and failure rates from introductory programming courses.

2.2.1 Encouraging Hands-On Practice and Experimentation

The literature on CS education embraces the notion that lots of hands-on practice and experimentation is especially important for novice programmers (Hassinen and Mäyrä 2006), because their knowledge of programming is not passively absorbed through texts and lectures, but rather actively constructed via their own practical experiences (Bruner 1990; Ben-Ari 1998; Huitt 2003).

Anderson et al (2007) encouraged students to actively participate during lectures through the use of laptop tablets giving students the opportunity to work with concrete examples, and the teachers the ability to view and discuss student annotations during the lectures.

2.2.2 Providing Multiple Entry Paths into Courses

Cohoon (2007) reports on the success of a pilot study carried out at the University of Virginia which offered multiple entry units into its CS course. While retaining the original unit offered, one additional unit was restricted to students with some prior programming experience and a second was for those without. Students elected to take one of the alternative units voluntarily. This had the effect of separating the students who were already comfortable with some aspects of programming from those who may be intimidated by displays of confidence in the knowledge of the course by others. The study resulted not only in comparable results in achievement by students from both alternative entries to those undertaking the original unit, but every student enrolled in one of the alternative units completed it and not one of them subsequently dropped out. Since the introduction of multiple entry options, there has also been an increase of 25% in enrolments for the course together with an increased selection of computing major, especially by females and minority students.

Another study at the University of Nebraska-Lincoln (Soh Leen-Kiat 2007) found similar success in having a placement examination for prospective students, and separating them into appropriate unit options, dependent on their results and their intended course path.

Offering CS0 level courses has had success in providing otherwise ill-prepared students with the necessary prerequisite knowledge better preparing them for CS1. Such elementary courses have been used to provide the foundations of computer science studies for
students from economically disadvantaged high schools which had provided no courses in computer science (Pearce 2011), and for those students swapping into computer science for the first time. Pearce's primary motivation for introducing the CS0 course was to generate interest from students in taking additional computer science courses, and found that CS1 enrolments were doubled after its introduction (Pearce 2011).

The prerequisite for entry into CS1 at Towson University of completion of a programming course in a high-level programming language was replaced with both an assessment test to determine preparedness or otherwise for CS1, and a CS0 course to provide the appropriate background for those students failing the entry test (Dierbach, Taylor et al. 2005). The entry test was designed to determine whether a student had the "minimal degree of programming reasoning ability expected for success in CS1" and consisted of 10 simple short answer or multiple choice questions written in pseudocode. The goal of CS0 was to "develop students' ability for algorithmic problem solving and program design" with the key measure of success being the students' final grades in CS1. The average grade for CS1 students who first completed the non-language-specific CS0 course was higher than students who had previously completed a programming course, even in the language used in CS1. These students commented that they had felt both "technically prepared and more confident" before starting their CS1 course.

2.2.3 Addressing Student Motivation and Engagement

One of the major reasons for students to drop out of IT courses was found to be motivation (Kinnunen and Malmi 2006). Much effort has been put into developing courses and tools which aim to motivate and captivate introductory programming students and make learning fun.

A gastronomic approach was taken by Davis & Rebelsky (2007) to engage and motivate students in a variety of topics during an introductory programming course. They demonstrated the development of an algorithm to instruct a computer to make a peanut butter sandwich. In this way, they introduced the students to the idea of algorithmic thinking. Learning for these students was fun, with stimulating exercises that they found amusing, as well as thought provoking.

Gonzalez (2006) found that introducing active and co-operative learning techniques resulted in pass rates rising from 44% to 70%, and attrition rates falling from 25% to 10%.
Much use has been made of games and puzzles as a way of making introductory programming courses more interesting. Pollard & Duval (2006) report on the successful use of "teaching techniques reminiscent of kindergarten: games, toys, play, and stories", all of which promote active learning, where students of varying levels of technical ability were motivated beyond grades and have fun during the learning process.

Many games and visual programming tools have been developed to help students learning programming, but with a definite bent towards student engagement, motivation and enjoyment. Some of these include:

- Parson's Programming Puzzles (Parsons and Haden 2006) — a tool designed to provide practice with basic programming concepts while making learning less tedious and more motivating. This is a pattern matching tool where students drag and drop elements of code to form a program solution.
- JavaTown (Feinberg 2007) — a visual programming environment, much like an object oriented version of the Little Man Computer (Englander 2000).
- PigWorld (Lister 2004) — which focuses on an objects-first approach to introductory programming, where algorithms emerge implicitly from the interactions between objects. PigWorld involves pigs who eat, sleep and have sex to reproduce.
- Alice (Carnegie Mellon 2007) — a narrative tool that supports programming by telling a story. Alice is designed to be a student's first exposure to object-oriented programming, learning while creating movies and games.

Curriculum design that focused on the interests of the students has had success in keeping students engaged in the material. Using image manipulation and creation has the advantage of providing students with immediate visual feedback and gives them a tangible outcome for their efforts (Corney, Teague et al. 2010; Dickson 2011).

In recent years with the proliferation of smart phones and mobile devices, ubiquitous computing and mobile applications development has exploded with a steady stream of new "apps" available for download (Rodrigues, Oliveira et al. 2010; Yan, Becker et al. 2011). iPhone and iPad development in particular has caught the attention of a new wave of potential entrepreneurs keen to be a part of the phenomena with development resources made readily available to them on the Internet (iOS_Dev_Center 2011). This has provided a unique opportunity for university students to learn about software development and to develop skills in an area to which they can directly relate and in which they have a keen
interest (Ivanov 2011; MIT 2011; Stanford 2011; Yan, Becker et al. 2011; QUT 2011(1); QUT 2011(2)).

The need to engage students was the motivation behind a successful breadth-first introductory CS course, where McFall and DeJongh (2011) designed and used a series of laboratory-based "authentic computing tasks". This was done to not only attract students to the course, but to engage them in its content because they were designed to be tasks that students were interested in knowing how to accomplish. McFall and DeJongh also highlighted via student feedback the value of hands-on implementation and experimentation to assist understanding:

"In the lecture the concept of [x] was just that, a concept. In the lab, it was made real. I discovered that [x] is a lot simpler than it seemed when we learned about it in the lecture".

2.2.4 Addressing Confidence, Enjoyment and Interest

It has been suggested that issues of lack of confidence, enjoyment and interest have a direct adverse effect on students' success in learning to program. Studies have found that women have less confidence than men when it comes to learning to program but it is our assertion that the issue of confidence and the potential for it to have a detrimental effect on learning outcomes is not gender specific.

A research project was undertaken at the Carnegie Mellon School of Computer Science (Fisher and Margolis 2002) to investigate and address the issues of the low percentage of women entering their undergraduate CS course. Twice as many women had been leaving IT courses than men. They found that women left as a result of lack of interest, resulting from loss of confidence in their studies. However, lack of confidence was not due to poor academic performance, but rather unfavourable comparisons with other students.

Carnegie Mellon addressed the issues identified from this research by promoting a broader perception of IT careers, rather than the stereotypical "hacker" or "geek" type roles. They introduced a more contextual approach to IT studies including interdisciplinary courses, human computer interaction, and involvement in non-profit organisations and community work which had greater appeal to female students. The research also identified a correlation between experience and confidence and as a result made allowances for students with less experience by introducing multiple entry options into the course. As a
result of this research and the action taken by the university to rectify the issues identified, female enrolments increased from seven to 42%.

In a more recent Australian study undertaken at the Swinburne University of Technology, Lewis, McKay et al (2006) reported on student views of teaching and learning. They found that women in the first year of ICT courses suffered from lack of confidence in what was viewed as a male dominated domain. Only 32% of women were prepared to approach academic staff for help, yet 53% of them were happy to ask their peers (compared to 55% and 66% respectively for men). A higher percentage of women than men also said they felt anxious about how they were coping with their studies, reported a heavier workload than expected and a lack of time to understand it. By comparison men reported too high an expectation by the lecturers and more pressure to perform well.

Katz, Allbritton et al (2006) found achievement as the determining factor on female attrition rates and that loss of interest can accompany loss of confidence. Gonzalez (2006) believed that confidence can play a significant role in the successful outcome of students learning to program. However, others report that lower confidence levels are not correlated to lower overall achievement (Murphy, McCauley et al. 2006).

Changes to computer science courses were suggested by Vilner and Zur (2006) to address both recruitment and retention issues. These suggestions include making the learning environment more hospitable and congruous with real life with collaborative learning, recruiting more female teaching staff and creating an atmosphere that supports development of positive attitudes towards computer science.

Addressing the "stereotype threat" and redefining the image of computing to increase the appeal to a more diverse range of students was proposed by Peckham, Stephenson et al (2007). They suggested this should be done at the institutional level, by the instructors themselves (the individual level) as well as reflected in classroom practices by the relevance to the particular learning styles and aptitudes of each student cohort.

Daly (2011) believes that confidence has the power to affect motivation, interest and achievement, and that students need to be challenged but not overwhelmed. For that reason Alice was used as the programming environment for an online introductory course. Daly found that the comfortable programming environment motivated students and improved their confidence and enjoyment of the learning process.
2.2.5 Using Tools and Tutorial Systems

An abundance of visualisation tools and tutorial systems have been developed to help students learn to program. Some of these include:

- **ELP** (Truong, Bancroft et al. 2003). QUT’s on-line environment for learning to program which offers a bank of fill-in-the-gap style exercises for a variety of programming languages. Students can save, compile and execute their attempts for each exercise; communicate with peers and tutors via an annotations tool, and have programs analysed in a simplified program learning environment.

- **jGrasp** (Cross, Hendrix et al. 2007). A "lightweight" IDE with an object viewer. It identifies data structures in Java code by examining the class structure based on a set of heuristics and generating a dynamic visualisation of that data structure.

- **Academic Java** (University of Exeter n.d.) is an on-line tutorial environment which allows compilation and execution of programs. It has the ability to step line by line through code with explanations and includes quizzes and tutorials and a quick reference guide.

- **JTF On-Line Tutorials** (Roberts 2006). The Java Task Force developed this set of library packages to address teaching issues in first year programming units. The packages include full interactive on-line tutorials to support teaching.

- **Programming Land** (Hill, Shanmugasundaram et al. 2006) is similar to an on-line text book which includes testing, communication and management features.

- **ILMDA** (Intelligent Learning Materials Delivery Agent) (Soh 2006) delivers learning materials based on the usage history of the learning materials, the students’ static background profile and the students’ dynamic activity profile.

- **Radenski** (2007) produced "digital study packs" including: self-guided, interactive and non-interactive laboratories; program samples; and quizzes. He introduces "abductive learning", a form of active learning that is targeted at abductive reasoning – which starts with a set of specific observations and then derives the most likely explanation of those observations. It is aimed at contemporary students who grew up browsing the Web and are less inclined to follow instructor’s presentations.

- **PeerWise** (Denny, Luxton-Reilly et al. 2008) is a system which requires students to create multiple choice questions as well as answer and evaluate those written by other students.

- **ViILLE** (Kaila, Rajala et al. 2010) is an online program visualisation tool designed for the creation of virtual programming courses and assessment items. This educational tool
supports multiple programming languages and allows teachers to annotate a program by inserting questions at specific locations in a program which may present a valuable teaching opportunity. ViLLE gathers valuable data relating to each student’s learning behaviour while interacting with the system.

Although the tools listed above potentially provide a great deal of support in terms of how and what material is presented to the student, they each tend to promote individual learning and allow little interaction and engagement with other students.

Tools have also been developed to analyse students’ learning progression and assess the quality and correctness of student programs:

- **Marmoset** (Spacco, Hovemeyer et al. 2006) — automated snapshot, submission and testing system.
- **Student progress monitoring tool** (Yoo, Yoo et al. 2006) — on-line tutoring system which includes testing and evaluation components. The student’s work is analysed and the results are presented to the teacher so that the student’s understanding can be systematically monitored.
- **Vee and Meyer** (2007) have reported on development of a tool using EiffelStudio which captures student compilation attempts which they refer to as "interaction logs", as well as gathering statistical data relating to those attempts with a view to developing an intelligent tutoring system.
- **Jadud** (2006) has also reported on development of a tool that explores compilation behaviour using BlueJ with particular emphasis on syntactical analysis. A "code browser" allows an instructor to read successive compilations of a program, and a visualisation captures the types and frequency of syntax errors encountered by novices in a single programming session. Sessions are then scored and quantitatively compared against one another.

### 2.3 Social and Cultural Barriers to Learning

The literature reveals that first year students in particular face not only cognitive challenges with complex topics like programming, but also a range of social and cultural issues during their transition into university. These barriers are likely to impede the students’ full potential from being realised or have more significant negative effects on their learning outcomes resulting in failure or withdrawal from the unit, or withdrawal from an IT degree entirely. Addressing some of these social and cultural barriers is the motivation for this
research and this section investigates how they manifest in first year programming students.

### 2.3.1 External Commitments

Many students find it necessary to take on part-time or even full-time work (James, Krause et al. 2010). Figure 2.1 shows the employment status of tertiary students in Australia in 1999. In that year, 57% of all students in higher education institutions also had a job, with 67% of part-time students working full-time and 42% of full-time students working part-time (Australian Bureau of Statistics 2000).

![Figure 2.1: Employment & Study Commitments](image)

More recent data from the Australian Bureau of Statistics shifted the focus to a workforce perspective. By 2010 in Australia, 14.5 million people in the 16-64 age bracket were in the workforce. Of those workers, 2.8 million (20%) were also engaged in education, 12% on a full-time basis (Australian Bureau of Statistics 2010). Figure 2.2 shows the breakdown of the Australian workforce in 2010 according to education status.
With an expected full-time study commitment of 48 hours or more per week at QUT and other similar universities, it becomes difficult to maintain employment, as well as to balance family commitments, social life, exercise and sleep. Increasingly, it seems, the corners that are cut are the hours dedicated to study, with attendance numbers dropping dramatically after the early weeks of semester, and students tending to leave assessment obligations to the last minute.

When the social, community and employment pressures are combined with a demanding subject like learning to program, the realisation that it takes a serious amount of time and effort to be successful often comes too late.

### 2.3.2 Cultural Perceptions

Programming is often perceived as a solitary occupation, one which is conducted in an environment that fosters competition and individualism (Werner, Hanks et al. 2004; Bagley and Chou 2007), rather than collaboration. This is often reinforced at university where introductory programming subjects’ assessment consists of individual assignments. Programming courses attract only a small proportion of female students and they have generally had less exposure to IT than their male counterparts (Cohoon 2002; Katz, Allbritton et al. 2006). Programming is seen as a competitive occupation whose model student is the stereotypical "geeky" young male, and this can lead initially to alienation, diminution of confidence and subsequent lack of interest for women (Fisher and Margolis 2002). Students are often misinformed about career opportunities and imagine the only
choice for a computer science graduate is to work as a programmer "... and study the computer for its own sake" (McFall and DeJongh 2011).

Academic ability would seem to have little influence on women’s attraction to and retention in programming courses. Women often perform well academically, yet perceive the programming environment as inhospitable, lacking social meaning and interaction which is incongruent with the real world. Women in IT are poorly represented at university which means there is less opportunity for support from female peers and possibly from role models in academic positions (Cohoon 2002; Vilner and Zur 2006). The attrition rate for women from computer science courses is generally higher than for men (Cohoon 1999; Fisher and Margolis 2002) and this has been evidenced at QUT with the dropout rate for domestic female students at 36%, compared to 23.5% for domestic male students in 2004.

For other minority groups historically underrepresented in computer science courses, there seems to be a positive correlation between their learning outcomes and how well they can communicate openly with their peers, with students’ social identities based on how well they fit into relationships with other students (Varma 2006).

So for female students and other minority groups in particular, learning to program may present further issues of a social and cultural nature. This may also be true for male students, although they tend to receive a higher level of support for entering and persisting in the field of computer science (Cohoon 2002).

2.3.3 Generation Y Culture

"The information technology revolution ... has been redefining the way we live, work, communicate, inform and entertain ourselves" (Mackay 2005).

It may be true that school leavers and younger students are resisting the way programming subjects are presented, being unwilling to acclimate to an environment that demands individual achievement and is devoid of the continuously interactive and social multimedia-rich world to which they have become so accustomed and reliant.

A 2005 US national survey of 8–18 year olds found that the amount of time each week young people spend on media is equivalent to more than a full time job (Rideout, Roberts et al. 2005). Rideout, Roberts et al report that today’s school children have become masters of multitasking, often using several forms of media, for example listening to music, having multiple MSN conversations and doing their homework simultaneously. While there
is now evidence that multitasking may not be a productive strategy (Rosen 2008) it certainly is part of the culture of Generation Y.

Robertson found a misalignment of expectations between faculty and academics delivering the introductory computing course, and students of that course (Roberson 2011). He attributes the differences in expectation to the generational differences between the staff (Baby Boomers and Gen-X) and the students (Gen-Y and Gen-Z) and after redesigning the course to more closely meet students’ expectations, has found that students’ reaction was overwhelmingly positive, and that retention into a subsequent course had increased significantly.

The constant interaction with peers during much of their leisure time via chats, email, SMS, etc. has become a significant part of the young university students' support structure for both their schooling and their social life. Generation Y students may not be able to adjust to a university environment that delivers linear content and demands individual, non-interactive study.

"The aging infrastructure and the lecture tradition of colleges and universities may not meet the expectations of students raised on the Internet and interactive games" (Oblinger 2003).

2.4 Collaboration
In the previous section, we described how social and cultural barriers manifest in first year programming students. Some of these barriers to learning can be addressed by focusing on the environment in which the students are expected to learn. Collaborative learning is introduced as a means of providing peer support for students, as well as developing the social interaction and encouragement especially beneficial for first year students in developing a sense of belonging and continued motivation. We detail the benefits of collaborative learning and carry that theme forward into collaborative programming. The focus is then drawn to the main theme of the thesis, pair-programming. Pair-programming uses the collaborative model and we describe its origins, protocols and how it is utilised in both industry and educational settings.

2.4.1 Benefits of Collaborative Learning
Collaborative learning is known to provide benefits to students including generating enhanced interest in the material, engagement in the learning environment, greater overall achievement and a more enjoyable learning experience.
Numerous studies (MacGregor 1990; Wilson, Hoskin et al. 1993; Yerion and Rinehart 1995; Williams and Kessler 2000; McKinney and Denton 2006) have found benefits to students learning collaboratively including:

- the synergistic nature of brainstorming and sharing of intellectual resources;
- support for monitoring the problem solving process by peer interaction which is conducive to successful performance;
- a positive effect of social interaction on cognitive growth and skill acquisition and transfer;
- a greater interest and sense of belonging;
- helping students apply algorithmic problem-solving techniques;
- deeper learning and higher retention;
- higher achievement and course success rates;
- developing skills wanted by industry; and
- enhanced confidence in the solution and enjoyment of the process.

Wilson, Hoskin et al (1993) highlight the value of integrating collaborative activities for first year university level computing courses. They also make assumptions that adults benefit in similar ways to evidence in the experimental literature on childhood learning which indicates that both high and low ability individuals benefit from self-explanation, verbalization and observation of their team mates.

A study by Gokhale (1995) examined the effectiveness of individual learning versus collaborative learning for students in a basic electronics course and found that collaborative learning fosters critical thinking through discussion.

Ben-Ari (1998) promotes collaboration between students in preference to individual work as it softens the brutality of the student’s interaction with the computer and facilitates the social interaction that is necessary for successful knowledge construction.

Supplemental Instruction (SI) aka Peer Assisted Study Scheme (PASS) is an academic assistance program that utilised recently high-achieving students to facilitate study sessions with current students in a collaborative, informal environment (UMKC SI 2007). The University of Missouri Kansas City – SI National Data – “National Supplemental Instruction Report” (2003) shows that rates of failure and poor marks of those that attend SI are significantly lower than those that do not attend SI.
McKinney and Denton (2006) experimented with collaborative learning in an introductory programming course at the School of Computer and Information Sciences, University of South Alabama. The students were exposed to team-based problem solving and pair-programming and the results provide confirmation of previous research that early use of collaborative learning leads to higher interest, higher retention, and higher academic performance in students. Other benefits of collaborative learning include developing skills wanted by industry, having fun, deeper learning and having a higher sense of belonging. These benefits were reported as being enjoyed by all students, but were particularly important for first year students who are at risk of leaving the discipline.

Gehringer, Deibel et al (2006) suggest extending collaborative learning beyond pair-programming and team projects, practices which they report the CS education community has "enthusiastically embraced". They suggest implementation of collaborative learning in the classroom to help boost the performance of at-risk students.

Students who were broken into numerous groupings were questioned by Bagley and Chou (2007) about collaboration. These groupings were: solo; pairs; mentor pairs (past student as mentor); and groups of 3. Students reportedly found collaboration most useful at the early stages (conceptualisation and formulating the problem requirements; designing each module) of more difficult tasks (at the point of having to transfer learning to a more complex program). Student grades were higher for pairs and (more significantly) mentor groups — regardless of gender. However, females rated the importance of collaboration higher than males. Those who worked in pairs and groups thought collaboration was important in increasing learning and motivation and those working solo rated it very low.

Collaborative learning establishes an environment conducive to learning and addresses the social and cultural barriers facing first year students and enhances their learning experience (Wilson, Hoskin et al. 1993; Gokhale 1995; Williams and Kessler 2000; McDowell, Werner et al. 2002; Gehringer, Deibel et al. 2006). Students benefit from 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).

Beck, Chizhik and McElroy (2005) conducted a controlled experiment in order to evaluate the effectiveness of having students role-play key Java concepts while focusing on problem solving and programming. For example, four students in a group would be assigned roles as follows:
- Variable Manager — responsible for keeping track of changing value of variables.
- Program Reader — reads the program and issues instructions to the Variable Manager.
- Method Executor — responsible for I/O operations.
- Facilitator — ensures each member stays in their assigned role.

The results of this experiment supported previous findings that cooperative learning improves learning outcomes and retention, particularly for ethnic minority and female students.

Kean University introduced Peer Led Team Learning as a method of small team collaborative learning into introductory computer science courses to aid in the retention of students in Science, Technology, Engineering and Mathematics majors (Stewart-Gardiner 2011). A clear majority of those engaged in the program improved their student ranking, with students giving positive feedback about the workshops, showing a link between interest and improved success. Failure and withdrawal of students persisting with the scheme were eliminated.

2.4.2 Collaborative Programming

Nosek (1998) conducted an experiment involving 15 full-time experienced system programmers from a program trading firm. Ten of the programmers formed five pairs who became the subjects of the experiment, while the remaining five were the control group of individual programmers. Nosek gives few details of the nature of the collaboration between each programmer in the pair, and simply describes pairs as "working jointly on the same algorithm and code". All 15 programmers were given 45 minutes to complete the same task. The task was to solve a challenging problem important to their organisation, in their own environments, using their own equipment. Programmers in this experiment admitted that they were sceptical of the value of collaborating on the same algorithm and code and did not expect the process to be enjoyable.

However, the results of this experiment showed that, compared to those working individually, programmers working in pairs:

- produced more readable and functional solutions; and
- expressed more enjoyment in the task and greater confidence in their solutions.

Although the pairs took less time to solve the problem than the individuals, Nosek's results in this regard are not statistically significant ($p > 0.05$). However, it is not unreasonable to
expect that given more experience with collaborative programming and time to develop a rapport with their partner, pairs may become even more time efficient.

Gallis, Arisholm et al (2003) summarise the types of collaboration that can occur in a programming environment:

- Individual programming — no collaboration, no shared workspace, limited communication with other programmers
- Partner programming — two programmers working on different tasks on different computers, but within the same general work area
- Team collocation — larger groups of programmers working in the same physical location, but using shared resources
- Pair-programming — two programmers working on the same task using one computer and keyboard

A study was conducted in a 2005 Advanced Java Programming Workshop course in the Computer Science Department of the Open University of Israel, where collaborative programming was investigated in the classroom. Benaya and Zur (2007) compared the results of projects where students voluntarily paired against those who worked on the projects by themselves. The students undertaking the projects were nearing the end of their degree and by prerequisite, had already successfully completed an Advanced Java Programming course and received a grade of 70 (out of 100) or above. The authors categorised the students in this study by their overall average grade, referring to "good" students as those with an average grade of 85 (out of 100) or above, and the others as "mediocre" students. They found that paired mediocre students tended to do much better in the project than mediocre students working solo. The outcome for good students working in pairs was not much different to those who worked solo. These results indicate that although there are obvious benefits of collaborative programming for some students, it is not necessarily something that should be implemented across the entire cohort. However, in terms of implementing collaborative programming in an introductory programming course, it would be difficult to determine exactly which of the students would fall into the "good" and "mediocre" ranges (or below), given that many of them would be in their first or second semester of university and have little or no academic history.
2.4.3 **Pair-Programming**

"Two programmers in tandem is not redundancy; it’s a direct route to greater efficiency and better quality" (Constantine (in Williams and Kessler 2000))

Pair-programming, which forms part of the eXtreme Programming (XP) concept (Beck 2005), is an example of collaboration being used successfully in professional software development environments. It is based on the collaboration of two software developers. One takes up the role of "driver", types the code, and addresses problems from a tactical point of view. The other becomes the "navigator" and thinks strategically, asks questions and watches for coding errors (Aiken 2004). The developers in the pair frequently swap roles to benefit from both experiences.

Beck (2005) explains pair-programming as part of the XP philosophy that holds coding as the key activity throughout a software project. One of the first large-scale IT projects that deployed XP principles was Chrysler’s Comprehensive Compensation (C3) project in 1996. Beck was tasked to resolve performance issues with Chrysler’s payroll system for its hundreds of thousands of employees (Haungs 2001). What evolved from that project was the set of principles that form the philosophy of XP, of which pair-programming forms part.

The basis of XP’s work ethic are as follows (Hislop, Lutz et al. 2002; Beck 2005):

- **Communication** — frequent high quality communication between team and clients
- **Simplicity** — a commitment to build the simplest system possible;
- **Feedback** — rapid integration, testing and estimations to provide constant feedback to engineers, programmers and clients;
- **Courage** — making tough decisions to ensure the best and simplest system design; and
- **Respect** — XP will only work if all members of the team care about the project and respect each other’s contribution;

with the basic XP activities being:

- **Coding** — the learning activity;
- **Testing** — integrated into the entire development phase. Each function must be completed before integrating the next;
- **Listening** — listening to clients to understand their requirements; and
- **Designing** — designing on the fly, a constant activity as the system evolves to incorporate new functions.
Pair-programming involves two programmers positioned comfortably side by side at the same computer, moving the keyboard and mouse between them as they change roles. The pair maintains a dialog while simultaneously programming (including analysis, design and testing of a project), in order to keep each other on task and to clarify ideas (Beck 2005).

Williams (2000) analogised the principles of pair-programming with child’s play along the lines of Fulghum's essay "All I Really Need to Know I Learned at Kindergarten" (1990) which suggests that the adult world would be a better place if only we practice ourselves what we preach to our children. These principles include:

- **Share everything** — equal ownership of and responsibility for the artifacts produced by the pair.
- **Play fair** — swap roles regularly and remain active and engaged in each role.
- **Don’t hit people** — keep each other on task, but do it in a non-violent manner.
- **Put things [negative thoughts] back where they belong** — have confidence in your own abilities and those of your partner.
- **Clean up your mess** — error detection is integrated into the development process as the presence of a partner helps with the cleaning up at the time the "mess" is made.
- **Don’t take things too seriously** — keep egos in check by accepting constructive criticism and by respecting opposing views of your partner.
- **Say you’re sorry when you hurt somebody** — effective communication is paramount. Adjust the workspace to allow for simultaneous viewing of the screen and for sharing the keyboard and mouse.
- **Wash your hands before you start** — clear away any skepticism and develop an expectation of success.
- **Flush** — either rewrite independent work or at least review it, as more bugs seem to come from independent rather than paired work.
- **Warm cookies and cold milk are good for you** — take breaks periodically.
- **Live a balanced life ...** — communicate with others often.
- **Take a nap** — work independently from time to time when it is more efficient to do so (e.g., simple, well-defined, rote coding).
- **When you go out into the world, watch out for traffic, hold hands and stick together** — work together for a single purpose with trust and loyalty.
• Be aware of wonder (and the power of two brains working together) — although both will probably have a large subset of knowledge and skills, but each partner will have some unique skills with which they can complement and amplify the other’s talents

2.4.3.1. Benefits

Experienced programmers who are conditioned to working by themselves often resist a transition to pair-programming, yet are able to make it with great success (Williams and Kessler 2000).

In an industry experiment reported by Jensen (2003) where programmers were paired according to the greatest difference in experience, productivity of paired teams improved by 127% and the error rate reduced by three orders of magnitude. Jensen listed the positive attributes of pair-programming over individual development:

• Brainstorming produced higher quality designs than achievable individually;
• Continuous design walkthroughs produced fewer errors;
• Improved focused energy;
• Improved mentoring: both elevating the junior’s skill and extending the craftsman’s skill quickly;
• Improved motivation; and
• Quicker problem isolation.

Jensen noted a negative aspect of pair-programming in this case was counter-productivity when the programmers were of the same experience and capability levels or had "prima donna" issues. Jensen suggested that an improvement in results may be achieved by having all the pairs work in a common area to increase interaction between the pairs.

Williams & Kessler (in Aiken 2004) summarise the primary benefits of pair-programming as:

• quality — pairs produce code with fewer defects;
• time — pairs produce higher-quality code in less time than individuals;
• morale — pair-programmers are happier programmers;
• trust and teamwork — pair-programmers get to know their team-mates much better, which builds trust and improves teamwork;
• knowledge transfer — pair-programmers, particularly those who don’t pair with the same person all the time, know more about the overall system; and
• enhanced learning — pairs continually learn by watching how their partners approach a task, how they use language capabilities, and how they use the development tools.

Williams & Kessler (in Aiken 2004) attribute the success of pair-programming to synergistic behaviours experienced including:

• pair pressure;
• pair negotiation;
• pair courage;
• pair reviews;
• pair debugging;
• pair learning; and
• pair trust.

Vanhanen and Lassenius (2007) carried out two surveys over a two year period in a department of a medium-sized Finnish software product company when it adopted pair-programming. The company had a development team of a few dozen and pair-programming was voluntary with 2–3 hour sessions recommended. Respondents agreed that the most effective pairs were those of dissimilar skill levels such as a junior and senior developer. Amongst other things, the study found positive effects on learning, quality, enjoyment of work and team spirit even though there was a somewhat higher level of effort than for solo work.

Begel and Nagappan (2008) surveyed a randomly selected ten percent of engineers at Microsoft and found that 22% of them either currently used or had experience with pair-programming. These engineers listed the top ten benefits of pair-programming as follows:

1. Fewer Bugs;
2. Spreads Code Understanding;
3. Higher Quality Code;
4. Can Learn from Partner;
5. Better Design;
6. Constant Code Reviews;
7. Two Heads are Better than One;
8. Creativity and Brainstorming;
9. Better Testing and Debugging; and
10. Improved Morale.
Interestingly, the Microsoft respondents listed three of the most important attributes of a programming partner as being "Complementary skills", "Flexibility" and "Good Communications", thus indicating that they placed less value on programming partners having similar or stronger skills. Complementary skills were said to be those that are "overlapping" rather than identical, and different perspectives provided by different backgrounds. In terms of skill levels, the engineers preferred their partners to have at least the same or better skills.

A four month ethnographic study of pair-programmers on two software development teams was undertaken in 2005. Chong and Hurlbutt (2007) found from their observations of the teams that when the two programmers had a similar level of expertise they engaged jointly in programmer activities. They did not witness a clear distinction between the roles of "driver" and "navigator", but rather a sharing of the responsibilities of both roles even though one programmer had control of the keyboard at any given moment. The pairs under observation were using computers with dual keyboard and mouse controls, so switching of control was rapid and tended to be frequent. When their skills were unmatched, the programmer with more expertise initially dominated the interaction, until a sufficient amount of shared expertise had been established. However, when the difference in expertise was large, the less knowledgeable programmer tended to become passive, reducing the benefits of the pairing.

Wray (2010) lists four mechanisms of successful pair-programming as:

1. Pair-programming Chat. This is the constant verbal interaction between the two programmers. Wray attributes much of the value in chatter to the phenomenon of simply verbalising one's issues as a means of stimulating clarity of thought process — even if no-one is listening. The effect is apparently enhanced if the audience is "perceived" as (but not necessarily is) a person of some expertise. There is added benefit in the listener offering intelligent prompting questions, even if they aren't particularly familiar with the topic.

2. Pair-programmers Notice More Detail. "Inattentional blindness" (or selective attention) means that when one programmer is focused on a task or an issue, they can easily miss something crucial that may be quite obvious to an onlooker. This is very clearly demonstrated by Chabris and Simons (2010) in their selective attention test. A video was shown of people tossing balls around, and those watching were asked to keep track of how many passes were made. Half of the audience did not notice a man in a
gorilla suit walk leisurely through the sphere of focus. The experiment revealed that we can be oblivious to what is staring us in the face. In terms of programming, this might be the difference between the programmer reading what the code ought to be and what is actually there.

3. Fighting Poor Practices. This brings in the consolidating effect of pair pressure and not wanting to let the other party down. The catch is of course that the programmers take ownership of the task, are adequately motivated to succeed, and actually reach agreement as to (amongst other things) the standard of their programming practices before they start work.

4. Sharing and Judging Expertise. Pair-programmers have the ability to recognise their own, and their partner’s expertise by sharing the development experience. Individual contribution and productivity of developers in a group environment are difficult to assess accurately, which is why group programming assessments are often avoided at universities. Having to demonstrate expertise, and articulate cognitive processes throughout development means that the two programmers are probably the best judges of the other’s abilities and productivity.

2.4.3.2. Pair-Programming in the Class Room

There is growing support for the use of pair-programming for learning (Werner, Hanks et al. 2004; Hanks 2006; McDowell, Werner et al. 2006; Mendes, Al-Fakhri et al. 2006).

Williams, Wiebe et al (2002) report educational benefits including

- superior results;
- increased satisfaction;
- reduced frustration;
- increased confidence on project results; and
- perseverance through the course.

Williams, Kessler et al (2000) conducted an experiment with senior software engineering students in order to validate pair-programming results observed in industry. Their quantitative results were similar to industry results, with pairs consistently passing more post-development test cases than individuals.

One deduction of this experiment was that a team’s productivity exceeded the sum of its individual’s productivity. Students who “jelled” in pairs after an initial adjustment period reported greater confidence in their work when their partner was observing them, and
because of their partner's vested interest in the task, felt they could be relied on. Likewise while taking the navigator role; they felt a responsibility to prevent any errors. Williams, Kessler et al also mention an on-line survey of professional pair-programmers, 96% of whom stated they enjoyed pair-programming more than working alone. 90% of the students in their experiment agreed. Similarly, improved confidence was reported by both the professional respondents and the students.

Bevan, Werner et al (2002) provide guidelines for using pair-programming in first year programming courses, which they say offers the added benefit of "breaking the pattern of conditioning students to work alone". They used pair-programming in their programming course to determine both its effectiveness as a teaching technique and its effect on student retention. A number of difficulties were encountered which mostly related to how well the two students worked together, including:

- disparity in the amount of experience and aptitude between students in a pair;
- students' understanding of pair-programming rules and guidelines and willingness to abide by them;
- student timetable scheduling and attendance; and
- variation in instructions given to students about the logistics of pair-programming and enforcement by instructors of its rules and guidelines.

The guidelines for effective pair-programming in the classroom that were compiled as a result of that study are:

- pair the students with similar skill levels;
- make laboratory sessions mandatory where students work as pairs;
- assign programming tasks that can be completed in the weekly laboratory session rather than expect them to meet outside contact hours, which can be logistically difficult;
- institute a coding standard that each student must adhere to; and
- create a pairing-oriented culture for both students and teaching staff that emphasises cooperation, mutual respect, and shared responsibility.

Nawrocki & Wojciechowski (2001) ran an experimental evaluation of pair-programming involving 21 students at the Poznan University of Technology. The experiment's objective was to compare pair-programming and other XP practices with the Personal Software Process (PSP). PSP (Humphrey 1997) is a framework to help software engineers apply
advanced engineering methods to their daily tasks and includes detailed estimating, planning and tracking methods. Five pairs were formed according to GPA (each pair having similar average GPAs) and became the experimental subjects using pair-programming. Five students used XP practices individually, and the six remaining students also worked individually using PSP. Students completed four programming assignments. The conclusions of these experiments were that pair-programming and test-centred XP programming by individuals were more time efficient than for individual programmers and that the students pair-programming produced more initially correct programs.

Muller and Tichy (2001) observed 12 CS graduate students in an XP course working on a traffic simulation project. The course involved students forming six pairs of their own choosing and meeting weekly on the project exercises. Students completed five questionnaires during the course, which provided feedback about their XP experiences during this course. Due to conflicting foci, one of the pairs did not enjoy the pair-programming experience which had a negative impact on their project design. There was mixed feedback from students about the type of tasks best suited to pair-programming, with some seeing little value in it for example when "rote coding" (writing simple blocks of code like getters and setters), while others preferred to pair permanently. Each pair had access to two monitors and commented that a single monitor would have been insufficient as one was used for searching and consulting documentation. 85% of students rated their enjoyment of pair-programming as average or better during the project, while 87% of them gave a similar rating after completion of the project. Students unanimously agreed that an advantage of pair-programming was learning from their partner, while 75% believed that it gave them greater confidence in their solution.

Gehringer (2003) conducted a pair-programming experiment in a senior/masters-level computer architecture class of 96 students in which programming was used, but not taught. According to a survey, they reported that significantly better grades were achieved when students paired for the most difficult of the three projects completed over the course of the teaching period. Students had the choice of pair-programming and the majority chose to do so for only two of the projects even though their feedback about pair-programming was positive in terms of co-operation and communication between the team members. On asking students "How was the experience of pair-programming?", Gehringer reports that students gave it a high rating. However, it is unclear from this particular survey question exactly what the students were rating. In terms of quantitative results of this experiment,
Gehringer remarked that even though the instructor reserved the right to assign different marks to each of paired students based on the students' feedback, there were in fact no cases where different grades were assigned. For one of the projects in this experiment, the most difficult of the three, paired students out-performed students working on their own. In the results for the other two projects, and the combined results of all three projects there was no statistical difference between the grades of the paired and solo workers. The grades achieved by students in this experiment are surprisingly high, with the mean grade being above 90% for each of the three projects. Gehringer remarked that the grades for programming projects did not vary much, but it is unclear if that was due to the students being of higher ability students, the non-challenging nature of the projects or a lenient marking system.

Cliburn's (2003) experiment in pair-programming suggests that the higher retention rate of paired students in introductory programming courses is because of an imbalance in skill levels within pairs, which enables the weaker student to attain a higher grade. In Cliburn's first small-scale experiment (N=27), the student pairs were first formed so that the members of a pair had different cultural and ethnic backgrounds, while the remaining students were grouped by disparate skill levels. Some students who attained good marks for paired assessment items performed poorly in exams, possibly indicating that weaker members of a pair had been "carried" by the stronger member.

Cliburn then performed a second (even smaller-scale, N=8) experiment, where pairings were made between students with similar course grades, and introduced peer evaluation as part of the assessment. Cliburn reported that these pairs seemed more cohesive, their final grades better, and there was a smaller difference between project and exam grades. However, the retention rate of students in this second experiment was lower. Cliburn concluded that the quality of the students who finished the course improved, and as a result the success rate in the subsequent course improved.

DeClue (2003) conducted another pair-programming experiment with students in a CS2 course which had them completing the first half of the course on solo tasks, and the second half in a pair. The students swapped partners every two weeks with one of the pair remaining with the current project, and the other moving to a completely unfamiliar code set. The belief was that this would have a positive effect on the production of quality code, and reinforce the importance of design documents, test plans, and documentation. Students completed peer evaluations and were also individually tested regarding the
specifics of their projects, both of which indicated how active a partner the student had been in the pair. Students in this experiment reported increased confidence in the readability of their code and thought that the quality of their code would not have been as high had they completed it by themselves. The students reported that they liked having another programmer with whom they could verbalise problem solving, and felt that fewer mistakes were made while pair-programming. The difficulties the students reported were scheduling time to work together and incongruent skill levels.

Gallis, Arisholm et al (2003) produced a framework for research into pair-programming. In this framework, in addition to control groups of solo programmers, there are also control groups of partnered programmers (two programmers working on separate tasks which will later be consolidated), and collocated programming teams. To determine the effectiveness of pair-programming, Gallis et al advocated that time, cost, quality, information and knowledge transfer, trust, morale and risk should be considered. The human factors considered most important by Gallis et al were personality, knowledge of pair-programming guidelines and roles, communication and swapping of partners.

McDowell & Hanks et al (2003) experimented with pair-programming in the classroom over three separate teaching periods, with computer science and computer engineering students who volunteered to take part. Students were given some explanation and background information to read, but as they were not supervised during pair-programming, it is unclear how closely the pairs followed pair-programming principles. Students were permitted to pair-program during development of assessment items. The conclusions drawn from this experiment supported other reports that pair-programming resulted in higher quality programs, and that there appeared to be no demonstrable disadvantages. One widely held belief is that with any collaborative assessment work, there is the potential for one student to assume a "parasitic" role by taking advantage of a partner who is prepared and able to "carry" them by doing the majority of the work, and consequently the student in such a passive role learns less. However, this study showed the "weaker" student in a pair did not perform significantly worse under examination than the "stronger" student, which is evidence against any parasitic affect, at least in this study. Data shows that paired students in this study were more likely to produce fully functioning programs for assessment than students who worked on their own, and therefore it is more likely that they learned more during a pair-programming experience.
In support of pair-programming, Nagappan, Williams et al (2003) concluded from experiments commenced in 2001 that pair-programming created a laboratory environment conducive to more advanced, active learning than traditional laboratories. They also remarked that both teaching staff and students agree that workshops implementing pair-programming tend to be more productive and less frustrating.

In a learning environment, Hanks & Werner et al (2004) suggest that pair-programming addresses social and cultural issues that students face including:

- confidence;
- inability to identify IT as "socially meaningful and interactive";
- inability to see IT as well-rounded or conducive to family life;
- image of IT as conducted in competitive rather than collaborative environment; and
- perception of IT as a solitary occupation.

Nicolescu & Plummer (2003) conducted an experiment in an advanced course on Distributed Computing in which the 240 enrolled students were supposed to complete three assignments and a midterm exam before being given the option to pair-program with a partner of their choice for a further two assignments. They gathered quantitative data in terms of assignment and exam marks, and surveyed the students at the time of the final assignment to determine their attitudes and perceptions of pair-programming and working individually. Those students who did pair-program had very positive attitudes towards it and thought that it was effective for assignments. They felt they learned more than they would have learned if they had worked alone and they also felt that pair-programming was more time efficient. The students who paired achieved significantly higher marks for their assignments than the solo workers. Students who chose to work alone instead of pairing performed better in both the midterm and final exams, leading to the conclusion by the authors that it seems that weaker students are more likely to choose to pair-program rather than work alone. It also shows that the pairing experience only had an effect on the work done during the pairing, rather than their ability to perform individually on exams. However, the pairing experience in this case was for only 24 days. Although the authors describe the final exam as broken into two parts, the first covering pre-pairing assignment topics and the other covering the paired assignment topics, a breakdown and comparison of students' marks for only the second part of the final exam is not evident and may have given a more accurate indication of any effect of pairing on exam performance.
Srikanth, Williams et al (2004) surveyed teaching staff and students who had been involved in pair-programming over the course of a semester. In this case, the students completed five assignments during the course of a semester and were assigned a new partner for each assignment. They found that although reassigning pairs throughout a semester required more effort from teaching staff, those staff valued pair rotation because it provided the opportunity for multiple peer evaluations and addressed issues of dysfunctional pairings. A significant number of students thought pair rotation was desirable, but did not believe that peer evaluation was a motivating factor for them. Students thought the value of rotation was in the exposure to a greater number of peers and through that greater exposure they learned more problem solving techniques. Apart from the issue of partner incompatibility, students saw the need to re-adjust to a new partner as a disadvantage.

Mendes, Al-Fakri et al (2005) conducted a controlled pair-programming experiment at the University of Auckland in 2004, involving 2nd year software development and design students. Students in a randomly selected sub-set of workshops were randomly assigned to different pairs every four weeks. This study showed a better rate of success for paired students than those who worked by themselves. It also reported an overwhelming number of students enjoyed the pair-programming experience. Mendes et al thought that the large percentage of Asian students in the experiment was a factor in the high class response to enjoyment, given their culturally based preference for collaboration rather than individualism and competitiveness. Students who worked during workshops in pairs performed better with individual assignment work and examination. These results support the use of pair-programming as an effective programming/design learning technique.

A year later, Mendes et al (2006) replicated their study of pair-programming. The replication confirmed pair-programming as an effective programming learning technique with improved quality of subsequent independent assignment work, examination scores and overall grade and reported improved enjoyment by the students involved.

Hanks (2006) reported specifically on student attitudes to pair-programming, which he found were mostly positive. In contrast to other reports, Hanks found that the more confident students liked pair-programming more than the less confident students, but like many other reports he found that pair-programming appeared to be particularly beneficial for women. Confidence and retention in computing-related majors increased more for women students who paired than it did for men.
Further support for pair-programming in the classroom comes from McDowell, Werner et al (2003; McDowell, Werner et al. 2006) where they found paired teams significantly outperformed individual programmers in terms of program functionality and readability. They reported greater student satisfaction with the problem-solving process, greater confidence in their solutions, and higher completion rates for programming assignments. The paired students produced significantly better programs than those who worked individually. The authors reported that paired students were significantly more likely than non-paired students to complete the course and consequently to pass the course.

Le Jeune (2006) studied students using eXtreme Programming practices in a capstone software engineering course. On a Likert scale of 1 (Agree) to 6 (Disagree), students rated their perception of pair-programming as follows:

- Easy to use: 1.83
- Works well to create required solutions: 2.17
- Produced higher quality than working alone: 2.22
- Enjoyed it more than working alone: 2.28
- The navigator's contributions are valuable: 1.78

These results reflect students' positive experience of pair-programming, even though before commencement of the study they had made negative comments about the concept. Frequent contributions by the navigators led to the students reporting a very strong positive attitude towards the navigator's contributions during software development.

When using pairs in a design class, Matzko and Davis (2006) observed that novice programming students benefited most from the pairing in the earlier design phase of development rather than when writing code. Students were exposed to a "more efficient and reliable programming method" (p. 129) rather than the students' more accustomed technique of trial and error. Matzko and Davis also reported that more experienced students learn equally as well whether paired or not paired.

Layman, Cornwell and Williams (2006) designed a lab-oriented course at North Carolina State University which emphasised agile processes including pair-programming. The success of this course in terms of positive student evaluations was attributed to the way agile processes appealed to a wider range of personality types and learning styles. Students in this course completed a number of assessment items in pairs, as well as one assessment item by themselves, so they were able to make a comparison of the two different learning
experiences. Even introverted students, who were energised by working alone, saw pair-programming as particularly beneficial for avoiding getting stuck on the program logic and control flow they had experienced as solo programmers. Students commented that they also found it easier to make design decisions when they had a partner working with them.

Williams (2006) requested the views of computing educators in a survey announced on the mailing list for ACM’s Special Interest Group on Computer Science Education (SIGCSE). Around 50 educators responded to the request. Respondents cited benefits of allowing students to work together and pair-program as:

- a reduction in student anxiety and frustration, more supportive study environment and more camaraderie in the classroom;
- students learned more, produced more and submitted higher quality assignments; and
- the educators’ workload decreased with fewer assignments to mark, fewer plagiarism cases, and less time consulting with students on small technical issues.

Given these reported benefits, Williams questioned the requirement in many courses for student programmers to work completely alone on assessment items to the extent of forbidding discussion about problem solving ideas or designs. Williams suggested that collaboration in the classroom may increase and broaden participation in computing fields, as well as give students a more realistic view of working in the real world.

There is still debate ensuing about the selection of pairs. Contrary to some of the previously mentioned research findings, Preston (2006) believes that pairing students of differing abilities and experiences is advantageous. Preston argues that the benefits include the fact that the inexperienced student has a mentor to guide them through particular concepts, and the experienced student’s own knowledge and understanding is reinforced by being put in a quasi-teaching role. Perhaps the students’ personality, learning style and motivation (amongst other things) are contributing factors, hence the option of self-selection is valid.

A study of 1350 students at North Carolina State University (Williams, Layman et al. 2006) was conducted to determine whether the likelihood of forming compatible pairs could be improved if selection was based on: personality type, learning style, skill level, programming self esteem, work ethic, or time management preference. This study found for CS1 students, the most important attribute for compatible pairings was a similar perceived technical competence.
A meta-analysis of studies on the effectiveness of pair-programming in both professional and educational settings (Dyba, Arisholm et al. 2007) concluded that pairs of programmers were more effective than individual programmers in terms of correctness when working on complex tasks. For novice programmers, there also seems to be a beneficial effect on program quality in working collaboratively on easy tasks.

In a small-scale experiment of five student programmers (Van Toll, Lee et al. 2007) one of the students was paired in turn with each of the other four students, who were of differing skill levels. Each pair worked on separate tasks using different programming languages. The student common to all pairs ("the common student") had a different level of skill for the programming language used in each of the four projects undertaken. The purpose of the exercise was to determine how the difference in skill contributes to the amount learned. In this qualitative research, Van Toll, Lee et al found that when the common student had less experience, he learned better collaboratively than from learning by reading a book. Syntax errors were caught early, and the pair pressure ensured both students stayed on task. When paired with a more experienced student, the constant questions from the common student extended the other student's knowledge and understanding of the language. However, the common student found that pairing with students of significantly less experience had a detrimental effect on the pair’s effectiveness. He became frustrated at the volume of questions and interruptions, and by the lesser experienced student’s declining interest in the project. Programming with similar skill levels produced no "mentoring" or "pair learning" situations, however there was evidence of significant pair problem-solving. Van Toll, Lee et al conclude that pair-programming was most effective for the learning of both weaker and stronger students when there was some difference (but not a significant difference) in skill levels between the pair of students.

Pair-programming in the learning environment addresses many of the issues that students struggle with when learning to program; not only cognitive issues, but also social and cultural issues. Students find programming in pairs creates a social rather than competitive environment which promotes interaction and lends twice the cognitive resources and an extra set of eyes to a programming exercise (Simon and Hanks 2007).

Chigona and Pollock (2008) experimented with a class of 32 students from an introductory programming course for Information Systems students, for two weeks of a 13 week semester. For the first week, half the students were paired, and the other half worked individually. For the second week of the experiment, the two halves swapped modes of
programming, so that all students experienced pair-programming. Before the experiment, students reported that they generally preferred working alone than in a team, but felt positive towards working in a pair-programming team. During the experiment, the average grade awarded to pairs was higher than those for individuals, although the authors did not make it clear whether the difference was statistically significant. The students agreed that pair-programming helped improve the quality of their work, and they were able to quickly resolve small technical issues within the pair, rather than having to seek help from teaching staff. Students also reported an improvement in enjoyment and productivity when they paired. Chigona and Pollock claimed that there seemed to be no notable transfer of knowledge between the pair because exam marks for pairs were not significantly higher than those of individuals. However, the similar results in exams for paired and non-paired students means it may also be concluded that the students’ learning was not compromised by an unequal contribution within pairs during the weekly exercises.

Williams, McCrickard et al (2008) reported on their use of pair-programming for two weeks of a Human-Computer Interaction (HCI) course involving 22 students. For these two weeks, the usual lecture and in-class activities were replaced by pair-programming sessions to implement a navigation interface for Tablet PCs. Pairs were reassigned for the second week at the end of which each individual was expected to submit a unique assignment. In this report, Williams, McCrickard et al expanded on the guidelines for implementing pair-programming in the classroom devised originally by Williams (2007). The original guidelines were based on the pair-programming experiences of over a thousand students at North Carolina State University over seven years. Their revised guidelines, which are similar to those of Bevan, Werner et al (2002) are as follows:

**Guideline 1:** Students need training in pair-programming in a supervised setting to experience the mechanics of successful pairing.

**Guideline 2:** Teaching staff must actively engage in the management of pair interactions.

**Guideline 3:** Strict attendance and tardy (sic) policies are necessary to protect students from a non-participatory partner.

**Guideline 4:** (When pair-programming outside a closed laboratory or classroom setting) instructors should provide a systematic mechanism for obtaining students’ feedback about their partners and must act upon the feedback when indications are a student is not being an equal participant.
Guideline 5: In each course, students should be evaluated on a balance of individual and collaborative work.

Guideline 6: When assigning pairs, instructors should attempt to maximise the chances students will work well together.

Guideline 7: Students should have different partners throughout the semester.

Guideline 8: Students must understand that problems with their partner must be surfaced (sic) immediately to give the instructor a chance to correct the situation.

Guideline 9: Pairs should be able to comfortably sit next to each other as they work, and both should have easy access to the monitor, mouse and keyboard.

Guideline 10: The programmers in a pair should be working towards a common goal.

Guideline 11: Teaching staff should encourage pairs to find answers on their own rather than providing them with answers.

Salleh et al (2009) considered the more "human" (p. 214) aspect of pair-programming in their study by investigating the effects of personality attributes on pair-programming. The main personality trait that Salleh et al focused upon was "conscientiousness", where a high level might be indicative of a student who was a high achiever, organised and thorough, while at the lower end a student would be seen to be disorganised and unprepared. Each student was ranked at one of three levels: low, medium or high. Pairs were then labelled as being either the "same" personality type (either conscientious or not), or of a "mixed" type. On average, paired students from a mixed personality group outperformed other pairs in individual tests. Salleh et al also found that differences in personality traits had no significant effect on individual performance in assignments for the students who had pair-programmed. Approximately 67% of the student subjects responded to a questionnaire about their perceptions of pair-programming. A very high proportion of responding students (89%) were satisfied with their partner, and reported a similar level of confidence to their partner's confidence for the solutions they jointly developed. Almost all (93%) of the responding students enjoyed the collaborative learning experience.

In a longitudinal study over several years, Braught et al (2010) experimented with students in an introductory computer science class, comparing academic outcomes for various types of students. The types of students under examination consisted of non-paired students,
those matched according to similar demonstrated abilities, and others in pairs that were chosen at random. Students in pairs completed weekly assessments collaboratively, while all students completed a number of written and programming exams. They concluded that the performance of less able students in individual programming tasks (the exams) may be improved by having them matched according to ability for the collaborative elements of the course. Although the comparison included non-paired and "random pairs", it would have been interesting to see the results of students paired according to opposing demonstrated abilities. There is little information in the paper about the spread of abilities in the random pairs, and the breakdown of pairs with similar and opposing abilities.

California State University proposed using a combination of pair-programming and peer-led team learning in closed laboratories after using each of these methods individually with limited success (Han and Beheshti 2010). Their experience with pair-programming proved ineffective because of students' lack of motivation, and reluctance to ask the laboratory assistants questions, as well as the laboratory assistants' inability to cope with a large number of pairs. Their trial of peer-led team learning as an additional resource seemed to be effective as an aid to learning but suffered poor participation rates from students who did not have the time for extra contact. Their proposal is for students to be paired according to their performance in class and peer leaders for the pairs chosen from an upper level class and trained on the mentoring program. The idea is for the experienced peers to mentor the pairs and keep them on task with the laboratory instructor available to assist both mentors and students should the need arise.

In summary, pair programming presents a model for students to develop programming skills in a collaborative environment. This section of the literature confirmed the benefits of collaborative learning which have already been discussed in previous sections, and identified some additional potential benefits specific to learning to program:

- [programming is] more productive, less frustrating and less stressful;
- early recovery from being 'stuck' on program logic, control flow and bugs;
- more efficient code is produced;
- more readable solutions are produced;
- higher completion rates of programming assignments;
- more enjoyment of the process (of learning to/developing program);
- greater confidence in program solution;
- encourages active learning and experimentation;
exposure (at an early stage) and adherence to efficient and reliable programming method;
• higher grades;
• higher completion rates; and
• improved image of programming and IT as a career path.

The studies which used pair programming as the collaborative learning tool identified some of the issues of establishing pair programming in the learning environment as follows:

• how pairs are chosen (self-selected or not);
• basis on which students are paired (level of experience; skill level; cultural and social demographic; timetabling);
• pair programming training (instructing students about the purpose, benefits, logistics and protocols of pair programming);
• how often pairs are reformed (if at all);
• how often the roles of driver and navigator are swapped; and
• adherence to pair programming protocols:
  • role swapping at regular intervals; and
  • maintaining open communication.

2.4.3.3. Distributed pair-programming

The feasibility of distributed pair-programming was investigated by Baheti (2002) who conducted an experiment involving North Carolina State University (NCSU) students undertaking an object-oriented programming team project. Baheti compared the productivity and project quality of students working in collocated, distributed, paired and unpaired groups. Distributed pairs were provided with desktop sharing software, called “Net Meeting”, together with audio communication equipment and instant messaging which provided an effective technical platform for remote teaming. Students pair-programming in a distributed environment produced equivalent grade programming projects as collocated paired and non-paired students, and were found to be as productive as students in the control groups. Baheti concluded that pair-programming in collocated and virtual teams were feasible ways to develop Object Oriented (OO) software and were comparable in terms of productivity and quality. Students involved in the experiment remarked that pair-programming in a distributed environment fostered teamwork and communication.
Another distributed pair-programming experiment was carried out by Canfora, Cimitile et al (2003) using volunteer software engineering students to determine the impact of distribution on the quality and productivity of pair-programming applied to maintenance tasks. Like Baheti’s experiment (2002), the distributed pairs used desktop sharing, with Net Meeting and instant messaging to communicate remotely, but Canfora et al’s students were not supplied with audio communication equipment. The results of this experiment contrast with the previous findings that working in a distributed environment fostered teamwork and communication. Canfora, Cimitile et al (2003) found communication to be the key factor for effective distributed pair-programming, and that when it is not well-enabled, students tend to stop collaborating and work solo, a phenomenon they refer to as dismissal. The findings of this study enumerate several reasons for communication breakdown, which can be overcome by providing the students with the following:

- sufficient instruction regarding the role of each member in the pair, and the accepted protocol for assuming and swapping roles;
- pairing students with similar skill levels;
- ample opportunity for paired students to work together and become familiar and comfortable with that working environment;
- opportunities for frequent brainstorming to consolidate and confirm common objectives;
- communication technology that closely resembles collocated dialogue (i.e. audio communication) to enable the students to continue development uninterrupted while in dialog, as well as instant messaging; and
- technology for version control of the pair’s developing project.

Hickey and Langton et al (2005) describe GREWPtool, which is a collaboration tool to allow non-collocated programming students synchronous control of the same code. The authors say this tool exploits the benefits of pair-programming, yet by providing synchronous access allows for a breakdown of enforcement of the distinct roles of the pair of programmers as would be the case with a collated pair sharing one terminal, keyboard and mouse. Their definition of "pair-programming" may be more accurately referred to as collaborative programming or partner programming. There is mention of "floor control" which indicates that one person (tutor or student) may take sole control of the editing window with the other/s in watch mode, but this feature is not described in detail and gives no clues as to how control is given/taken and therefore how swapping roles in a pair would be handled. In any case, GREWPtool provides for collaboration between two or more students, often by
a whole class of students and their tutor. Communication between the users is via an instant messaging type of window, public and private browser frames and the editing frame. The tool displays a tabbed window, with each tab representing a unique group of collaborators, with each user in that group assigned a colour for their cursor and for their last 20 characters of text entered into the editor. There is also a convenient feature called “VCR” which provides a "recording" of activities of all the user’s actions, allowing for a rewind and replay of edits. Although the tool was predominantly tested in a classroom setting, a number of users contributed from remote locations. The authors identify lack of audio communication as a disadvantage for those students using the application remotely.

Goldman (2010) called for innovative changes to driver/navigator protocol due to inconsistent evidence of its structure in practice. They proposed a system of pairs in roles based on test driven development. Using a shared environment but with different views, one programmer writes the tests, the other implements the system. Failed tests are returned to the implementer for modification, and development continues with passing of unit tests back and forth. The dependencies between the roles continue until mutual satisfaction of the product and the pair moves on to the next task. Professional software developers used a web based development environment to evaluate this "side by side" programming. The programmers were said to have worked along a spectrum of solo to pair-programming and using their supported environment, it would be possible to do this in both collocated and distributed environments. Unequal workloads were mentioned, as was the limited swapping of roles which may not particularly suit the learning environment where students benefit particularly from seeing and hearing the other programmer and being actively involved in both roles on the same task.
### Summary of Empirical Research

Table 2.1 below is a summary of formalised empirical research into pair-programming to date:

<table>
<thead>
<tr>
<th>Authors</th>
<th>Year of publication</th>
<th>Type of study</th>
<th>Subjects</th>
<th>n</th>
<th>Task</th>
<th>Duration</th>
<th>SP?</th>
<th>Independent variable</th>
<th>Main dependent variables (metrics)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nosek</td>
<td>(1998)</td>
<td>Ex</td>
<td>Pros</td>
<td>15</td>
<td>One challenging task — real world</td>
<td>45 mins only</td>
<td>Yes</td>
<td>5 solo → 5 pairs (small collaborative groups act pair-programming)</td>
<td>readable and functional solutions; confidence in solution and enjoyment of process</td>
</tr>
<tr>
<td>Williams</td>
<td>(2000)</td>
<td>CS Students</td>
<td>20</td>
<td>11 weeks</td>
<td>Web site development using CSP</td>
<td>Yes</td>
<td>Evaluation of PP to gather experience with the process</td>
<td>View of programming using CSP (qualitative assessment)</td>
<td></td>
</tr>
<tr>
<td>Williams, Kessler, Cunningham, &amp; Jeffries</td>
<td>(2000)</td>
<td>Ex Students</td>
<td>41</td>
<td>4 assignments</td>
<td>6 weeks</td>
<td>Yes</td>
<td>PSP (13 solo) versus CSP (14 pairs)</td>
<td>Total work time, productivity and programming quality</td>
<td></td>
</tr>
<tr>
<td>Haungs</td>
<td>(2001)</td>
<td>CS Pros</td>
<td>2</td>
<td>C3 system</td>
<td>Yes</td>
<td>Investigate use of PP</td>
<td>Information and knowledge transfer (qualitative assessment)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nawrocki and Wojciechowski</td>
<td>(2001)</td>
<td>Ex Students</td>
<td>21</td>
<td>4 programs</td>
<td>N/A</td>
<td>PSP (6), XP with PP (5 pairs), and XP with individual progr (5)</td>
<td>Time; quality (lines of code); number of resubmissions due to defects</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Muller and Tichy</td>
<td>(2001)</td>
<td>CS Students</td>
<td>12</td>
<td>Software tasks</td>
<td>11 weeks</td>
<td>Evaluation of XP (including PP) to gather experience with the process</td>
<td>Information and knowledge transfer; morale</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Baheti</td>
<td>(2002)</td>
<td>Ex Students</td>
<td>132</td>
<td>OD Software Engineering project</td>
<td>Semester</td>
<td>4 groups of students: Collocated and paired Distributed and paired Collocated and unpaired Distributed and unpaired</td>
<td>Comparison of grades, productivity and quality</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bevan et al</td>
<td>(2002)</td>
<td>CS Students</td>
<td>9</td>
<td>9 assignments</td>
<td>10 weeks</td>
<td>Yes</td>
<td>Evaluation of PP’s effectiveness as a teaching technique and effect on student retention</td>
<td>Total work time, satisfaction with partner, amount of time worked alone</td>
<td></td>
</tr>
<tr>
<td>McDowell et al</td>
<td>(2002)</td>
<td>Ex Students</td>
<td>313</td>
<td>5 assignments</td>
<td>2 sessions of academic year</td>
<td>Yes</td>
<td>Solo (141) versus pairs (86 pairs)</td>
<td>Quality — score on programming assignment (functionality and readability), Learning effect — score on final exam</td>
<td></td>
</tr>
<tr>
<td>Canfora, Cimitile &amp; Visaggio</td>
<td>(2003)</td>
<td>Ex Students</td>
<td>16</td>
<td>Software Engineering maintenance task</td>
<td>Distributed pairs → collocated pairs</td>
<td>Quality and time</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Study</td>
<td>Year</td>
<td>Type</td>
<td>Students</td>
<td>Assignments</td>
<td>Duration</td>
<td>Match</td>
<td>Pairing Methodology</td>
<td>Methodology</td>
<td>Findings and Implications</td>
</tr>
<tr>
<td>------------------</td>
<td>------</td>
<td>------</td>
<td>----------</td>
<td>-------------</td>
<td>----------</td>
<td>-------</td>
<td>---------------------</td>
<td>-------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>Gallis et al.</td>
<td>2003</td>
<td>CS</td>
<td>Pros</td>
<td>4</td>
<td>Project coding task</td>
<td>Est 5 mths</td>
<td>Partner programming (1 pair) versus pair programming (1 pair)</td>
<td>Information and knowledge transfer, morale</td>
<td></td>
</tr>
<tr>
<td>Gehringer</td>
<td>2003</td>
<td>Ex</td>
<td>Students</td>
<td>96</td>
<td>3 assignments</td>
<td>Semester</td>
<td>No</td>
<td>Computer architecture class: Solo (19,29,17) versus pairs (41, 23, 28 pairs)</td>
<td>Quality — score on assignment</td>
</tr>
<tr>
<td>Cliburn</td>
<td>2003</td>
<td>CS</td>
<td>Students</td>
<td>14, 8, 17</td>
<td>5 assignments</td>
<td>Semester</td>
<td>Yes</td>
<td>All paired</td>
<td>Learning effect — final course grade, PP enjoyment (qualitative survey)</td>
</tr>
<tr>
<td>McDowell et. Al</td>
<td>2003</td>
<td>Ex</td>
<td>Students</td>
<td>95, 19, 102</td>
<td>3 months</td>
<td>Semester</td>
<td>Yes</td>
<td>Solo (47, 5, 44) versus pairs (22, 7, 29)</td>
<td>Qualitative assessment of program quality, Learning effect — score on final exam</td>
</tr>
<tr>
<td>Nagappan et al</td>
<td>2003</td>
<td>Ex</td>
<td>Students</td>
<td>199, 502</td>
<td>3 programming projects</td>
<td>Semester</td>
<td>No</td>
<td>Solo (69, 102) versus pairs (22, 140 pairs)</td>
<td>Learning effect — course, assignments, and exam scores, attitude towards PP (qualitative)</td>
</tr>
<tr>
<td>DeClue</td>
<td>2003</td>
<td>Ex</td>
<td>Students</td>
<td>24</td>
<td>6 assignments</td>
<td>Semester</td>
<td>No</td>
<td>Solo (22) and same 22 paired for 6 weeks</td>
<td>Student attitude towards PP and role of PP on code/design quality (qualitative survey)</td>
</tr>
<tr>
<td>McDowell et al</td>
<td>2003</td>
<td>Ex</td>
<td>Students</td>
<td>552</td>
<td>5 or 4 assignments</td>
<td>1 year: (4 sections, 3 months each)</td>
<td>Yes</td>
<td>Solo (148) in Spring section pairs (202 pairs) in Fall and Winter sections</td>
<td>Student success — final course grade and exam attendance. Gender completion rate — final course grade, course performance — score on final exam and program quality</td>
</tr>
<tr>
<td>Jensen</td>
<td>2003</td>
<td>Ex</td>
<td>Pros</td>
<td>10</td>
<td>Project</td>
<td>Yes</td>
<td>All paired with others of opposing skill level</td>
<td>Productivity; Error rate</td>
<td></td>
</tr>
<tr>
<td>VanDeGrift</td>
<td>2004</td>
<td>Ex; Su</td>
<td>Students</td>
<td>546</td>
<td>3 projects and corresponding reports</td>
<td>Semester</td>
<td>Yes</td>
<td>All paired</td>
<td>Students processes — subjective assessment of reports, students perceptions on PP and written reports (survey)</td>
</tr>
<tr>
<td>Nicolescu &amp; Plummer</td>
<td>2003</td>
<td>Ex; Su</td>
<td>Students</td>
<td>240</td>
<td>2 assignments</td>
<td>4 weeks</td>
<td>Mostly</td>
<td>Paired -v- non-paired</td>
<td>Student perceptions on pair-programming v. individual work Midterm and final exam marks -v- assignment marks</td>
</tr>
<tr>
<td>Srikanth et al</td>
<td>2004</td>
<td>Ex</td>
<td>Students</td>
<td>270, 140</td>
<td>5 assignments</td>
<td>2 semesters</td>
<td>No</td>
<td>All paired</td>
<td>Students and teachers perceptions on pair rotation and students perceptions on peer evaluation (all surveys)</td>
</tr>
<tr>
<td>Katira et al</td>
<td>2004</td>
<td>Ex</td>
<td>Students</td>
<td>564</td>
<td>Up to 6 assignments</td>
<td>2 semesters</td>
<td>Yes</td>
<td>All paired</td>
<td>Understand compatibility of pairs — survey</td>
</tr>
<tr>
<td>Hanks et al</td>
<td>2004</td>
<td>Ex</td>
<td>Students</td>
<td>112, 50</td>
<td>5 assignments</td>
<td>Semester</td>
<td>Solo (112) and paired (25 pairs) at different semesters</td>
<td>Program quality — complexity, length, subjective metrics, students confidence and satisfaction</td>
<td></td>
</tr>
<tr>
<td>Original Sources</td>
<td>SP?</td>
<td>Su</td>
<td>Ex</td>
<td>Students</td>
<td>Project Course</td>
<td>Semester</td>
<td>No</td>
<td>Evaluation of XP (including PP)</td>
<td>Qualitative Questionnaire Responses</td>
</tr>
<tr>
<td>------------------</td>
<td>-----</td>
<td>----</td>
<td>----</td>
<td>----------</td>
<td>----------------</td>
<td>----------</td>
<td>----</td>
<td>--------------------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>Le Jeune (2006)</td>
<td></td>
<td></td>
<td>Ex</td>
<td>18</td>
<td>Project course</td>
<td>Semester</td>
<td>No</td>
<td>Individual -v- paired programming tasks</td>
<td>Qualitative questionnaire responses</td>
</tr>
<tr>
<td>Mendes et al (2005)</td>
<td>Ex</td>
<td>Students</td>
<td>300</td>
<td>Workshop exercises</td>
<td>Semester</td>
<td>No</td>
<td>Evaluation of pair-programming for learning; (quantitative) grades; (qualitative) student attitudes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vanhanen &amp; Lassenius (2007)</td>
<td>Su</td>
<td>Pros</td>
<td>15-22</td>
<td>Project</td>
<td>&gt;10 hrs</td>
<td>No</td>
<td>Voluntary pairing within development team of a few dozen</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chong &amp; Hurlbut (2007)</td>
<td>Obs</td>
<td>Pros</td>
<td>9, 10</td>
<td>Software development</td>
<td>40 hrs total over 4 month period</td>
<td>No</td>
<td>XP pairs negotiated daily; dual keyboards; dual mice; rapid and frequent switching of control</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Van Toll, Lee et al (2007)</td>
<td>Ex</td>
<td>Students</td>
<td>5</td>
<td>4 programming tasks</td>
<td>Yes</td>
<td>1 student paired with each of the other 4 of different experience levels</td>
<td>Measure of effectiveness of PP with different experience levels (qualitative)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chigona &amp; Pollock (2008)</td>
<td>Ex</td>
<td>Students</td>
<td>32</td>
<td>Programming exercises</td>
<td>2 weeks (4 &amp; 9 of 13 wk semester)</td>
<td>16 paired; 16 individual for 1st week; reversed in 2nd</td>
<td>Gauge quality (quantitative); experiences and attitudes towards PP (qualitative)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Williams, Crickard et al (2007, 2008)</td>
<td>Ex</td>
<td>Students</td>
<td>22</td>
<td>Assignment</td>
<td>2 weeks of a 15 week semester</td>
<td>No</td>
<td>HCI course implementing pair-programming</td>
<td>Qualitative evaluation of effectiveness of guidelines previously developed from anecdotal experience</td>
<td></td>
</tr>
<tr>
<td>Salleh et al (2009)</td>
<td>Ex</td>
<td>Students</td>
<td>52</td>
<td>Intro programming</td>
<td>Summer school</td>
<td>Yes</td>
<td>Personality traits</td>
<td>Effects of personality attributes on PP’s effectiveness; student satisfaction (qualitative survey)</td>
<td></td>
</tr>
<tr>
<td>Braught (2010)</td>
<td>Long. Ex</td>
<td>Students</td>
<td>259</td>
<td>CS1 course</td>
<td>13 groups of 1 Semester course</td>
<td>No</td>
<td>Pair selection according to demonstrated ability</td>
<td>Improved performance on individual programming tasks for students who were paired by similar ability</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.1: Summary of Empirical Pair-Programming Research

SP? = same partner?  
Su = survey  
Ex = experiments  
Obs = ethnographic observation  
CS = case study  

Original sources: (Gallis, Arisholm et al. 2003; Mendes, Al-Fakhri et al. 2005)  

An empty cell means that the information was either unclear or not evident from the reference material.
2.5 Summary

There is widespread interest in addressing introductory programming failure rates and attrition from computer science courses, and some of the issues confronting teaching institutions and the pedagogical choices made to address these issues have been discussed. Often, in search of a silver bullet, the choices made by educators have focused on programming language, paradigm and course content. This literature review then concentrated on more “people focused” approaches to programming pedagogy. The possibility of non-cognitive, social and cultural barriers to learning programming was discussed with the first year student in focus. This led to the bulk of the investigation into previous related work which centres on collaborative learning and how it addresses student learning issues summarised in Table 2.2 below.

<table>
<thead>
<tr>
<th>Issue</th>
<th>How Addressed with Collaborative Learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cognitive development — knowledge building</td>
<td>Peer support; verbalisation, collaborative problem solving</td>
</tr>
<tr>
<td>Perception of male domination</td>
<td>Potential to pair women; increase confidence</td>
</tr>
<tr>
<td>Perception of solitary occupation</td>
<td>Socially interactive learning environment</td>
</tr>
<tr>
<td>Socially immature or introverted students</td>
<td>Shy students are encouraged to interact and develop a rapport with at least one other student</td>
</tr>
<tr>
<td>Extroverted students who see university as an opportunity to socialise</td>
<td>Students are encouraged to interact with others as they feel comfortable; many strong friendships and study relationships continued long after the collaborative learning experiment finished</td>
</tr>
<tr>
<td>Lack of confidence</td>
<td>Students who fear being disadvantaged because of no prior knowledge can be mentored and encouraged by a peer of superior knowledge; or two such inexperienced students can develop pair courage because they are not working alone</td>
</tr>
<tr>
<td>Over confidence</td>
<td>Highly confident students are given the opportunity to demonstrate their prowess and it is self-evident if their confidence is unfounded</td>
</tr>
<tr>
<td>Apathy and/or lack of motivation due to prior knowledge</td>
<td>Experienced students get the opportunity to mentor their peer if experience levels are unbalanced — reinforcing and extending their knowledge by “teaching”</td>
</tr>
<tr>
<td>Programming has a “geeky” image</td>
<td>Promotes a social, not competitive environment;</td>
</tr>
<tr>
<td>Lack of social interaction</td>
<td>Provides a socially interactive learning environment</td>
</tr>
<tr>
<td>Generation-Y</td>
<td>Collaboration encourages use of various Web 2.0 tools (i.e., those which facilitate on-line interactive information sharing and collaboration) in order to communicate regularly and efficiently</td>
</tr>
<tr>
<td>Abductive thinkers</td>
<td>Investigative learning; and experimental discovery, interactive learning</td>
</tr>
<tr>
<td>School leavers — time to party</td>
<td>Peer pressure; peer commitment</td>
</tr>
<tr>
<td>Commitments outside university leading to constraints on time</td>
<td>Assistance of collaborative learning tools; Web 2.0 communication tools</td>
</tr>
</tbody>
</table>
Table 2.2: Issues Addressed with Collaborative Learning

<table>
<thead>
<tr>
<th>Issue</th>
<th>How Addressed with Collaborative Learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cultural/religious differences</td>
<td>Peers provide intra and inter-cultural support</td>
</tr>
<tr>
<td>Language/communication difficulties</td>
<td>Peers communicate more easily with each other, and pool communication resources when communicating with others in order to understand what initially may not have been clear</td>
</tr>
</tbody>
</table>

Providing a collaborative learning environment which mimics pair-programming enables students to share mentoring roles, reflect on their own and each other’s work, and share the learning experience during problem solving and program development. With collaboration able to provide such significant benefits and programming students being more likely to engage in a collaborative environment, pair-programming has the potential to positively affect course success rates, reduce attrition and in turn attract more interest from potential students, especially women and minority groups.

The collaborative nature of pair-programming in particular demonstrates to students that software development need not be the competitive, socially isolating activity that they imagined. Learning in a collaborative environment can help address confidence issues and make learning programming a less intimidating, and more enjoyable activity.

This research will now contribute to the body of existing literature in support of pair-programming as a successful learning environment. We will gather evidence that pair-programming as a pedagogical approach for learning to program improves learning outcomes.

Qualitative data will be collected in order to learn more about student cohorts and their perceptions, attitudes and motivations for studying programming to help us understand the "people issues" associated with learning to program.

We will conduct in-class experiments with introductory programming students over several semesters to quantify the effect of pair-programming on final grades.

We will confirm the positive effect of pair-programming on student learning outcomes by improving the three key areas of student perceptions of:

- confidence in being able to successfully learn to program;
- difficulty of learning to program; and
- enjoyment of learning to program.
Quantitative analysis of student final grades for introductory programming units using pair-programming will be made in order to corroborate existing literature and to confirm our hypothesis that such a learning environment addresses social and cultural issues for students learning to program.

2.5.1 Research Opportunity

What is the best approach to use when implementing pair-programming as a strategy for teaching and learning?

Firstly, we should measure students’ perceptions of their confidence in programming, how difficult they find it, and how much they enjoy it. Measuring these perceptions continuously over a teaching semester will help us to understand what the social and cultural barriers to learning to program are, and how and when they may best be addressed.

For pair-programming, ensuring that each student is paired with the most suitable of peers in order to maximise their opportunities for efficient collaboration is a task that requires some modelling and guidelines. The basis for pair selection should be grounded in empirical knowledge rather than simply left to chance or determined by teaching staff based on student data. Pair-programming over the course of a semester would require that students are paired very early in semester, for example in Week 1. For introductory programming units which are normally run in the first semester of the first year of the degree, this is a time when teaching staff would normally have very little working knowledge (if any) of their students. There must be criteria developed, beyond just academic history and assumed personality types, to assist this selection process. It is with these selection criteria that students will be able to make informed decisions about the type of student with whom they would most effectively pair.

Little is also documented in the current literature about the process of enforcing the pair-programming protocol during its implementation. How do teaching staff ensure that student pairs use single terminals and are collocated in a practical way in order to pair-program efficiently? How can pairs be encouraged to swap roles at appropriate intervals and maintain constant communication?

Very few empirical studies using pair-programming in introductory programming units have been conducted over successive teaching semesters. What are called for are longitudinal experiments using the same teaching units, applying the same treatment of pairing to a
proportion of the student cohort, with final grades compared to those of a control group who work individually.

Multiple cycles of pair-programming under similar teaching conditions would be invaluable to help determine the most effective ways to develop a learning environment which uses pair-programming. We conduct our research for this Thesis using QUT’s introductory programming units to replicate pair-programming experiments over several semesters and to case study novice programmers. Student results from those multiple cycles of the same teaching unit will be analysed to determine the impact pair-programming has on student learning outcomes.

With multiple iterations of the same teaching unit, each experiment cycle presents an opportunity to refine the process of pair selection as well as modification and adherence to pair-programming protocols which develop a unique pair-programming environment for students to best learn.

This research will track students’ perceptions to studying programming in the key areas of confidence, enjoyment and difficulty, then implement a series of pair-programming experiments over a number of semesters to address any negative perceptions. As well as quantitative analysis of the experiment results, students will be asked to reflect on their experiences of pair-programming to help us learn about the effect that this type of collaborative learning has on social and cultural barriers to learning.

The remainder of this thesis documents the investigations made into the “people issues” affecting learning to program, and the development of a unique learning environment using pair-programming in the classroom and its effect on student learning outcomes. The following chapter describes how these investigations were carried out over numerous teaching periods.
Chapter 3. Methodology

3.1 Introduction

The literature identified universal difficulties in recruiting, engaging and retaining IT students at universities. The underlying motivation for this research was to mitigate the difficulties of learning to program. Some of those difficulties relate to issues of confidence, enjoyment and perceived difficulty of programming.

Collaborative learning is known to be a valuable pedagogical tool as it addresses issues such as confidence and enjoyment, and has the potential to improve student learning outcomes. According to the literature, pair-programming provides a sound method for collaboratively learning to program, the benefits of which have been described in the previous chapter. In Table 3.1 below, we use the key findings from the literature review in Chapter 2 to conduct further research into pair-programming in the classroom on the following bases:

<table>
<thead>
<tr>
<th>Issue Identified in Literature</th>
<th>How Issue Was Addressed</th>
</tr>
</thead>
<tbody>
<tr>
<td>How pairs are chosen (self-selected or not)</td>
<td>Student pairs to be self-selected — based on students' own criteria, and so they retain some ownership of and responsibility for the partnership</td>
</tr>
<tr>
<td>Basis on which students are paired (level of experience; skill level; cultural and social demographic; timetabling)</td>
<td>Students choose partners according to their own priorities, with guidance (in terms of experience [not very disparate skill levels], timetabling, motivation and goals)</td>
</tr>
<tr>
<td>Pair programming training (instructing students about the purpose, benefits, logistics and protocols of pair programming)</td>
<td>Training and adequate background information be provided to students and tutors at beginning of semester, in order that they are fully aware of the purpose, benefits, logistics and protocols of pair programming</td>
</tr>
<tr>
<td>How often pairs are reformed (if at all)</td>
<td>For logistical reasons as well as continuity, pairs remain together for the duration of the semester — unless problems arise that threaten either student's learning potential</td>
</tr>
<tr>
<td>How often the roles of driver and navigator are swapped</td>
<td>Tutors enforce regular swapping of roles every 15 mins or so, or at more appropriate times that lessen disruption and avoid discontinuity (eg after small exercises or related block of code finalised)</td>
</tr>
<tr>
<td>Adherence to pair programming protocols:</td>
<td>Tutors remind students to swap roles at appropriate times, and encourage open and active communication between students in and out of the classroom</td>
</tr>
</tbody>
</table>
- role swapping at regular intervals
- maintaining open communication

Table 3.1: Establishing Logistics of Pair-Programming based on Key Findings
This chapter describes the design and methodology adopted for this project, using multiple research methods. We discuss the three key issues of perceived confidence, difficulty and enjoyment and implementation of classroom experiments using pair-programming as the modus operandi for collaborative learning. These empirical studies were complemented with qualitative surveys, case studies and observations of students learning in pairs.

3.2 Multi Method

This research aimed to better understand the barriers that students face while learning to program including issues relating to their perceptions of programming, their confidence at being able to successfully learn to program and their enjoyment of programming. After designing an appropriate collaborative learning environment for introductory programming students, we aimed to quantify the effect that pair-programming had on students' ability to learn to program. This research therefore employed a multi-method approach (Bassey 1999; Dick 2000; Yin 2003) or triangulation of data collection including both quantitative and qualitative research methods.

3.2.1 Causal Studies

Experimental pairing of students was used to facilitate collaborative learning and the influence this had on their learning was observed. Quantitative analysis included comparison of final results between the paired and non-paired students from the same unit. The initial pair-programming experiment was conducted over two separate semesters for the first programming unit, and one semester for the second programming unit. After redevelopment of the first programming unit, the experiment was replicated and data gathered over a period of three consecutive semesters. Further details about the units partaking in the experiments can be found in Section 3.3.1 below.

3.2.2 Qualitative Research

Qualitative data was collected over one semester including:

- a retrospective survey ("Then and Now Survey"), at weeks 1 and 8 of semester, of students' perceptions in terms of their level of:
  - confidence at being able to successfully learn to program;
  - enjoyment of learning to program; and
  - difficulty in learning to program.
- case studies of a small number of volunteer students from both units involving structured weekly interviews; and
• weekly reflection surveys (perception of individual topic/s) carried out during lectures. Qualitative data was collected over several semesters including:

• students’ own reflection of their experience over the semester they were learning to program.

(A detailed description of the introductory programming units at QUT can be found in Section 3.3.1).

3.3 Data Collection

This research commenced in Semester 1, 2007 when the author was part of the teaching team for both the introductory programming units at QUT. Details of each of the data collection periods follow in Section 3.3.1.

This research spanned several teaching semesters (see Figure 3.1 below), providing an opportunity for a longitudinal study of using pair-programming in a learning environment. First year introductory programming units at QUT were used in this study, with the main aim of grounding the learning environment in collaborative peer interactions.

![Figure 3.1: Data Collection Periods](image)

The initial stages involved qualitative analysis of students' perceptions of their own confidence and expectations about the level of difficulty and enjoyment of programming, and their receptiveness or otherwise to a collaborative learning environment. The aim was to determine the social and cultural background of students and to obtain an understanding of their perceptions of factors affecting their learning of programming including difficulty, confidence and enjoyment. This data was gathered by using a variety of surveys and questionnaires which will be discussed in more detail later in this chapter.

The feedback given by students in the initial stage of analysis led to development of in-class experiments to measure the effectiveness of learning to program using pair-programming. These experiments involved a select number of workshops for each of the units and allowed for the observation of students while their learning took place.
A comparison of final grades between paired and non-paired students was used to measure the success or otherwise of the pair-programming experiments. In order for the distinction between these two groups of students to be as clear as possible, the pair-programming protocols needed to be adhered to and teaching staff were set the task of informing and enforcing the logistics of this type of collaboration in the classroom.

3.3.1 Quantitative Data: In-Class Experimental Studies
Multiple iterations of in-class experiments were conducted from which quantitative data was used to measure the effect of pair-programming on student learning. The following are descriptions of each of the teaching units at QUT involved in this research, each of which represents 25% of a full-time study workload and 12 credit points towards a degree.

3.3.1.1. ITB001 — Problem Solving & Programming (Semester 2, 2007)
ITB001 was the first introductory programming unit that students completed at QUT, usually in the first semester of their first year.

Scheme was used as the programming language, and the unit covered expressions, procedures, repetition, data abstraction, I/O, message passing and memory, and basic object oriented programming.

All assessment items were for individual completion and consisted of three programming assignments making up 50% of the grade. A final end-of-semester exam accounted for the other 50%.

An extract from QUT’s published unit outline for ITB001 for Semester 2, 2007 can be found in Appendix A.

Lectures were instruction based and followed by workshops which were held every week for two hours. Workshops were conducted in flat rooms where students were expected to problem solve and design solutions to given programming tasks. The students who were the subjects of the pair-programming experiment were those who had allocated to workshops facilitated by the author. After the workshops students were encouraged to attend drop-in sessions in computer laboratories to implement the solutions in Scheme. These sessions were facilitated by student teaching assistants who were able to offer some guidance in implementing the solutions developed during the workshops, but who did not enforce pair-programming protocols.
3.3.1.2. ITB003 — Object Oriented Programming (Semester 2, 2007)

ITB003 was the second introductory programming unit offered by QUT, normally undertaken by students in their second semester of their first year.

The programming language used was C#, and the unit covered language syntax, test plans, OO design and structure, GUI development, data structures, I/O and exception handling.

All assessment items were for individual completion and consisted of a programming project making up 30% of the grade, workshop programming questions for 10% and end-of-semester short review questions for another 10%. A final end-of-semester exam accounted for the other 50%.

An extract from QUT’s published unit outline for ITB003 for Semester 2, 2007 can be found in Appendix B.

Lectures were instruction based, and were followed by workshops which were held every week for two hours, and were conducted in computer laboratories where students were expected to design and implement code solutions to given programming tasks. The students who were the subjects of the pair-programming experiment were those who had allocated to workshops facilitated by the author.

3.3.1.3. ITB001 — Problem Solving & Programming (Semester 1, 2008)

This iteration of ITB001 was a modification of the unit described in Section 3.3.1.1, having been redeveloped using the Python programming language.

The unit covered expressions and assignment, functions, conditions, list processing, iteration and recursion, error detection and handling, program efficiency and correctness, file I/O, abstract data types and an introduction to object oriented programming.

All assessment items were for individual completion and consisted of three programming assignments making up 50% of the grade and on-line quizzes for 10%. A final end-of-semester exam accounted for the other 40%.

An extract from QUT’s published unit outline for ITB001 for Semester 1, 2008 can be found at Appendix C.

Lectures were instruction based, and were followed by workshops which were held every week for two hours and conducted in computer laboratories where students were expected
to design and implement code solutions to given programming tasks. The students who were the subjects of the pair-programming experiment were those who had allocated to workshops facilitated by the author.

3.3.1.4. INB104 — Building IT Systems (Semester 1, 2009)

INB104 replaced ITB001 as the first introductory programming unit at QUT, normally to be completed in the first semester of the first year of study. Python continued to be used for the programming language, but in line with a course restructure, this unit was redeveloped into a breadth-first introduction to the basic building blocks of IT systems, including programming, databases and Web development. It offered students a taste of different technologies, which helped them make informed decisions about subject and stream selection for the remainder of their course. “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.” (Corney, Teague et al. 2010)

The unit covered an introduction to IT systems and the System Life Cycle, expressions, functions, lists, conditionals, iteration and recursion, introduction to databases and SQL tables and queries, web servers, HTTP, HTML, top down design, abstract data types and objects.

Assessment consisted of a portfolio of projects that students submitted in three stages which contributed 60% of the grade. This portfolio was developed in pairs as a joint submission where each member of the pair was awarded the same mark. Other assessment items were on an individual basis and included weekly on-line quizzes for 10% and a reflective journal at the end of semester worth a further 30%.

An extract from QUT’s published unit outline for INB104 for Semester 1, 2009 can be found at Appendix D.

Lectures were demonstration based with an emphasis on designing before coding. Lectures incorporated development of a solution from analysis phase through to design and implementation, and further code samples were supplied as additional examples. Workshops were held every week for two hours, and were conducted in computer laboratories where students were expected to design and implement code solutions to given programming tasks. All students undertaking this unit in this semester were subjects of the pair-programming experiment.
Chapter 3: Methodology

3.3.1.5. **INB104 — Building IT Systems (Semester 2, 2009)**

This iteration of INB104 was only slightly modified from the previous semester. The course content remained unchanged.

In the previous semester, none of the first semester units in the Bachelor of IT course included a final exam in their assessment. For this second iteration of INB104 a final exam was introduced as the primary means of assessing students' individual, working understanding of the concepts covered in the unit. This sole first semester exam served as a useful comparative aid to students' collaborative submissions and helped to further eliminate the opportunities for plagiarism and/or parasitic behaviour. Plagiarism had typically been an issue for this subject's predecessor, with normally between five and ten cases being detected. There was a reduction in plagiarism for INB104 assignment to only two cases (Corney, Teague et al. 2010). Students were given ample time to complete the exam and this served to ease students into the university examination process that would become more intense and challenging in future years.

Assessment was changed to a portfolio of projects that students jointly submitted in two stages which contributed 50% of the grade. Other assessment items were on an individual basis and included weekly on-line quizzes for 10%, a reflective journal at the end of semester worth a further 10% and the final exam worth 30%.

An extract from QUT's published unit outline for INB104 for Semester 2, 2009 can be found at Appendix D.

3.3.1.6. **INB104 — Building IT Systems (Semester 1, 2010)**

This iteration of INB104 was unaltered from the previous semester. An extract from QUT’s published unit outline for INB104 for Semester 1, 2010 can be found at Appendix D.
3.3.1.7. **Summary of Teaching Units**

The following Table 3.2 highlights the similarities and differences between the teaching units involved in this research that were described in the previous sections.

<table>
<thead>
<tr>
<th>Sem</th>
<th>Unit</th>
<th>Language</th>
<th>Content</th>
<th>Workshop</th>
<th>Assessment</th>
</tr>
</thead>
<tbody>
<tr>
<td>2, 2007</td>
<td>ITB001</td>
<td>Scheme</td>
<td>Introductory programming - first unit</td>
<td>2 hrs flat room</td>
<td>Individual + exam</td>
</tr>
<tr>
<td>1, 2008</td>
<td>ITB003</td>
<td>C#</td>
<td>Introductory programming - second unit</td>
<td>2 hrs laboratory</td>
<td>Individual + exam</td>
</tr>
<tr>
<td>1, 2008</td>
<td>ITB001</td>
<td>Python</td>
<td>Introductory programming - first unit</td>
<td>2 hrs laboratory</td>
<td>Individual + exam</td>
</tr>
<tr>
<td>1, 2009</td>
<td>INB104</td>
<td>Python</td>
<td>Introductory programming, databases and web</td>
<td>2 hrs laboratory</td>
<td>Paired assignments + individual quizzes and journal + no exam</td>
</tr>
<tr>
<td>2, 2009</td>
<td>INB104</td>
<td>Python</td>
<td>Introductory programming, databases and web</td>
<td>2 hrs laboratory</td>
<td>Paired assignments + individual quizzes and journal + exam</td>
</tr>
<tr>
<td>1, 2010</td>
<td>INB104</td>
<td>Python</td>
<td>Introductory programming, databases and web</td>
<td>2 hrs laboratory</td>
<td>Paired assignments + individual quizzes and journal + exam</td>
</tr>
</tbody>
</table>

Table 3.2: Comparison of Teaching Units

3.3.1.8. **Workshop Pairing**

The background literature provided the principles for establishing pair-programming experiments in the classroom. The basic protocols of pair-programming were adhered to as closely as possible. Teaching staff and student subjects were instructed on these protocols, the importance of adherence to them, and the potential benefits and issues likely to be encountered. As the teaching units involved in this experiment were managed by a number of different unit co-ordinators and facilitated by changing academic teaching teams, the application of this experiment varied between semesters. The pair-programming scope remained under the ultimate control of the individual unit-co-ordinator.

During the first semester of the experiment in ITB001, as the workshops were conducted in flat rooms, pair-programming was used in the design phase only, but with the principles of pair-programming enforced with pen and paper in lieu of computer equipment.

Table 3.3 that follows outlines the basic protocols that were established for the classroom:
Pairing

Students in the workshops involved in the pair-programming experiment were not given a choice whether to pair or not. Students chose their own partners. Teaching staff offered some assistance in the event of one student later dropping the subject or the partnership deteriorating for any number of reasons. Students were put in a group of three rather than work alone. In a small number of cases where students refused to collaborate, they were identified as working solo and moved to the control group in terms of the experimental data.

Duration

All pairs were established in the first week of semester, with the intention that they remain in the same pair for the duration of the semester.

Tasks

Pairs were required to work together during allocated weekly workshops on specific programming exercises. The replicated experiments allowed students to also work on and submit assignment tasks as a pair.

Roles

Each student in the pair took on a different role for each exercise (or in the case of larger exercises, the roles were to be swapped at regular intervals e.g., each 15 minutes or so):

a) the “driver” took control of the pen/keyboard: e.g., writing/typing and refining the stepwise development of an algorithm; typing in the translation to code; and executing and debugging the code;

b) the “observer” was responsible for thinking strategically, asking questions, watching for errors, suggesting alternatives, and providing technical input;

c) tutors enforced the separate roles, reminded students to swap roles at appropriate times and encouraged intensive and continuous interaction between the students in a pair.

Outside class

Pairs were encouraged to meet/make contact at other times to complete workshop exercises for example:

a) face to face;

b) directly before or after the workshop each week;

c) at another time on or off campus that suited both students;

d) remotely — by arranging a mutually agreeable time to communicate each week via:

(i) MSN (or other public chat tool);

(ii) on-line chat on the university’s learning management system, Blackboard, using breakout for private pair discussions; or

(iii) telephone/VOIP (e.g., Skype).

Table 3.3: Pair-programming Protocols in Classroom

Information sheets were distributed to students at the beginning of each semester of the experiments which reiterated the background information on collaborative learning discussed by the teaching staff in the first workshop. The sheets included justification for the use of collaborative learning, and the protocols and logistics involved with pair-programming. See Appendix E for an example information sheet.

Initially, students started the semester with a brief introduction to other students in the class. The class would gather in a large circle and each student would have a turn introducing themselves to the group and talking a little about their background and the
course they were enrolled in. After hearing everyone's story, students were given a little

time to choose a pair partner.

As it became apparent from observations and feedback of the importance of choosing
compatible partners, more onus was placed on students being able to make informed
decisions about who they would work with for the entire semester. More time was
allocated in the first week of semester to having students become familiar with their peers.

Ice-breakers were introduced where students had to write one lie and two truths about
themselves and stick them on a post-it note to the front of their shirt. They would then
move around the room trying to guess what each other's lie was. This resulted in a much
more interactive and interesting workshop to start the semester. However, the details
forthcoming from these ice-breakers did not necessarily provide enough specific
information about a student in order for their peers to accurately identify them as a
compatible partner.

Subsequent semesters saw ice-breaker guidelines being more specific about the types of
information for students to share. According to feedback from previous semesters, the
most significant issue of compatibility for pairs was the difference in skill levels. Other
issues reported were those of motivation and timetabling. So in the latter semesters of the
experiment the focus of the ice-breakers was on disseminating information about these
three key areas of experience, motivation and timetabling. Students were set to task
"sorting" themselves in three different ways, and then talking to other students they found
themselves with.

Firstly the students sorted in order of previous experience — from none or very little, to
some experience programming at industry level. Students were then able to talk to others
who were near their own level of experience and chatted about what they had already
done, and what they expected from this unit.

Then they sorted themselves into courses they were enrolled in, so they could meet others
who most likely had the same study timetable, and were interested in the same subjects.
Discussions at this time included timetabling and external commitments which may impose
restrictions on their ability to meet regularly. The last sorting was into home address
postcodes, in order to find others who lived in a similar or close area. Discussions included
travel times and other geographical issues that may prevent students meeting in person
easily and regularly.
After these discussions students had a much better idea of who they may be able to efficiently pair with, including personalities and demographics like age, culture and gender.

After the final instructions by the tutor, students were asked to choose a student with whom to pair. In the latter iterations of the experiment, very few students needed assistance in pairing, as they seemed to have a very good idea of who they wanted to be with. Once they had paired, they were instructed to agree on meeting times outside workshops and exchange contact details. Students were then told that they were now responsible for not only their own learning, but had a stake in their partner’s learning too. Students were told they were expected to be in contact with their partner several times a week, and that they should know when their partner wasn’t going to be showing up for workshops, and why not.

Because students were working in pairs, it was necessary to provide a style guide which ensured that the same conventions were used by both students, and that the final solution was consistently formatted and commented. This coding style guide was introduced early in semester and enforced throughout semester including all assessment items for which there was a component of marks.

The initial pair-programming experiments ran for two semesters before the introductory programming unit was redeveloped as a result of a course redesign in 2009. The redevelopment included the incorporation of pair-programming for all workshops. Students were required to work as pairs during all workshop activities and make submissions in pairs for the major assessment items. The unit content was also changed significantly from pure programming to a breadth-first introduction to the three technologies of programming, databases and web development. Detailed information about the motivation for change and the resultant impact made can be found in Corney, Teague and Thomas (2010).

Student results were gathered at the end of each semester for each of the teaching units involved in the experiments. An Analysis of Variance statistical test (Stockburger 1996) was used to determined the statistical difference between grades for students paired and students who were not paired. For the purpose of this analysis, students were categorised as "paired" if they attended more than half the workshops for the unit (that is, six or more), and actively worked with another student (or two) using the pair-programming protocols. The results of this analysis are detailed in Chapter 4.
3.3.2 Qualitative Data

To investigate the perceptions of introductory programming students, a number of qualitative data collections took place with a view to recording the key perceptions of confidence, difficulty and enjoyment of programming experienced at various periods throughout the semester.

3.3.2.1. Then & Now Survey

At Week 8 of Semester 1, 2007, 163 ITB001 students were asked to quantify their perception of programming by rating how difficult they were finding it, their current level of enjoyment, and their confidence at being able to complete it. By way of comparison, students were also asked to reflect back on their perception of the programming unit at the beginning of semester and rate their confidence, enjoyment and perceived difficulty of programming before they had actually started the unit.

Students were also questioned in this survey about their perception of learning in a collaborative environment, and (amongst other things) how such an environment might affect their confidence, enjoyment and difficulty of learning to program. The instrument for this survey is included in Appendix F.

This retrospective reflection gave students the opportunity to quantify any attitude change to the unit during the first 8 weeks of semester. For example, a student may have thought that the content was fairly difficult in Week 1 and had given it a 4 out of 5 rating. In Week 8 (after introduction of recursion and higher order procedures, etc.) that same student may in hindsight realise that their initial perception of programming was rudimentary and now not have the option of indicating how much more difficult they were finding it (e.g., off the scale at 9 out of 5 in comparison). That is, of course, if they remember how they rated it 7 weeks ago. There is some evidence to suggest that retrospective surveys may be more accurate than the more traditional pre and post approach, as “… response shift bias is avoided because participants are rating themselves with a single frame of reference on both the post-test and retrospective pre-test” (Pratt, McGuigan et al. 2000).

It should be noted that the students surveyed were those who attended the lecture in Week 8. Lectures and workshops normally suffer a steady decline in attendance (see Section 1.1.2), and an assumption could be made that those students surveyed perhaps represent the more motivated of the student cohort.
Each rating given by the students in the survey for the three categories was made on a scale of 1 to 5 as follows:

- **difficulty**: 1 = very easy, 5 = very difficult
- **enjoyment**: 1 = not enjoyable, 5 = very enjoyable
- **confidence**: 1 = no confidence, 5 = very confident

The results of the Then & Now Survey can be found in Section 4.2.1.

### 3.3.2.2. Case Studies

Case studies were conducted during Semester 2, 2007 in order to gain a better understanding of the students enrolled in our introductory programming units. Under investigation were student perceptions to and difficulties with studying programming, as well as their approach and attitude to learning. Case studies are an appropriate strategy for considering these types of queries (Yin 2003).

Interviews were conducted on a weekly basis involving students who had volunteered from either of QUT’s first year programming units in that semester. The units involved are detailed in Section 3.3.1. There were 13 ITB001 students involved in these case studies, and three ITB003 students. From these 16 students, eight turned up every week of semester.

A profile was initially taken of each student, and the instrument at Appendix G shows the demographic details recorded for each student.

Students were asked similar questions each week relating to how difficult or otherwise they were finding their current enjoyment and confidence levels. Students graded each of these perceptions on a Likert scale of 1 (low) to 5 (high). The students involved in the case studies were also asked how much time they actually spent on the unit each week\(^2\).

Appendix H documents the questions posed to students each week. Apart from these set questions, there was time each week for students to discuss whatever they wanted, and they were given the opportunity to set the direction and agenda of the interviews. Sometimes they took the opportunity to ask about programming ideas and a reinforcement of their understanding of the weekly topic. Other times were used to talk about life as a student in general and to develop a sense of trust and rapport between the interviewee and interviewer.

---

\(^2\) For each 12 credit point unit at QUT, students are expected to dedicate a total of 12 hours study each week including lectures, workshops and private study.
Chapter 3: Methodology  

A selection of the case studies are summarised in Section 4.2.2 including a profile of each student, their attitudes to study and their perception of confidence, motivation and enjoyment of programming throughout the semester.

The final grade awarded to each student (on a scale of 1 to 7) is included in the results of these studies in Section 4.2.2.

3.3.2.3. Weekly Surveys

In Semester 2, 2007, ITB001 students attending lectures were asked to complete a short survey each week relating to the work they had completed the previous week for this unit. The aim of these surveys was to allow us to follow the same cohort of students through the semester, measuring their perceptions of confidence, difficulty and enjoyment and gaining some insight into the "people issues" that may be negatively affecting their learning, as well as getting a fair indication of their study habits.

These surveys were completed anonymously, and to make the task of distributing and collecting the surveys each week less onerous and time-consuming, it was carried out when most students were together: at the lecture. For this reason, the perceptions of individual students were not able to be tracked throughout the semester. Instead, we were able to calculate the average in each of the key areas for the entire cohort of students. For the same reason, we were not able to compare paired and unpaired students’ perceptions from these records.

The questions, the same each week, required the students to quantify:

- how confident they were with the previous week's topics;
- how difficult they found those topics; and
- how much they enjoyed those topics.

Students were asked to rate each of these three key areas from 1 (low) to 5 (high). This gave us the opportunity to determine any correlation between perceived confidence and enjoyment levels as student progressed through more difficult programming topics. This information was necessary in order to develop a learning environment that addresses any negative perceptions and improves learning outcome.

Other questions were posed relating to how well students remembered and understood the topics. They were also asked how much time they dedicated to studying for this unit.
and how they used that study time. At QUT, the expectation is that students spend approximately 12 hours per week for each 12 credit point subject. (Each of the units involved in this research were 12 credit point subjects.) Generally, students were expected to attend the formal contact time involved in lectures and workshops and make up the remainder of the 12 hours in self-directed study which may include working on set exercises, assessment items, readings, peer mentoring programs and informal discussions with other students. The details provided by the students gave us some useful information about their motivation and study ethic as well as the opportunity for timely reflection on their progress each week.

Appendix I records the survey instrument used. Section 4.2.3 summarises the results of the weekly surveys.

### 3.4 Summary

This chapter described the surveys undertaken to collect evidence of students’ perceptions of confidence, difficulty and enjoyment of programming as they progressed through a semester of introductory programming. It also outlined the multiple iterations of an experiment of pair-programming in the classroom implemented to address negative student perceptions identified in the surveys. The experiments were used to measure the effect of this type of collaborative learning environment on learning outcomes.

The qualitative and quantitative results from the surveys and experiments form the basis of Chapter 4 which follows.
Chapter 4. Results

4.1 Introduction

Chapter 3 outlined the experimental approach used in this research and highlighted how it was carried out along two main axes. Students were surveyed in various ways to determine the impact of social and cultural experiences on their learning. In particular, perceptions of confidence, enjoyment and difficulty were studied, and a summary of findings can be found in Sections 4.2.1 and 4.2.3. Individual students were studied in depth to gain an understanding of how their background impacted on their learning. The outcomes of these case studies are given in Section 4.2.2.

Multiple iterations of using pair-programming in the classroom were implemented in a longitudinal manner using introductory programming units at QUT. Each implementation was used to feed into the subsequent iterations to improve the pair-programming approach used in the units. A comparison of results was made between paired and non-paired students to quantify the effect that pair-programming had on their learning. These results are given in Section 4.3.

4.2 Social and Cultural Experiences

4.2.1 Then & Now Survey

In the first week of Semester 1, 2007, students attending a workshop for QUT’s introductory programming unit, ITB001, were asked to share a little of their background with the class. It was evident that this student cohort had a diverse range of experiences with computers and programming. These ranged from basic exposure to computers in non-IT subjects at school to many years of experience as a result of keen personal interest and/or part-time job. Hobby and job related experiences seemed to be predominantly hardware related, with just a few students noting exposure to programming outside school.

A large group of the students aspired to careers in the gaming industry and in particular animation and graphics, while others simply enrolled in the IT degree because that is where they excelled at high school.

In Week 8 of Semester 1, 2007 (before collaborative learning was introduced into QUT’s programming units) students quantified their perception of programming by rating how
difficult they were finding it, their current level of enjoyment, and their confidence at being able to complete it. By way of comparison, they were also asked to reflect on their perception of the unit at the beginning of semester and rate the same perceptions before they had actually started the programming unit.

As students progressed through the university’s first programming unit, 49% of them found it *more difficult* than they expected. See Figure 4.1 below.

![Figure 4.1: Change in Student Perception of Unit Difficulty](image1)

In terms of enjoyment of the unit, 38% of students found the unit *less enjoyable* than they had anticipated it would be before they had actually started it. See Figure 4.2 below.

![Figure 4.2: Change in Student Perception of Enjoyment of Unit](image2)
The confidence levels of the same group of students fell during the first eight weeks of the unit, with 45% being less confident in being able to successfully complete it than they were at the beginning of semester. See Figure 4.3 below.

As the surveys were anonymous (as explained in Section 3.3.2), there is no way to determine if it were the same students who changed their perceptions on all three aspects.

With 85% of the students surveyed being under 26 years of age, this group is most likely to have had exposure to computers and IT at school and at home and therefore to be in a position to know a little of what to expect in a programming unit at university.

Paired t-tests were conducted on the changes in perception in terms of difficulty, enjoyment and confidence detailed in Figure 4.1, Figure 4.2 and Figure 4.3 above.

The null and alternative hypotheses are as follows:

$H_0$: the mean difference between the paired observations is zero

$H_1$: the mean difference between the paired observations is not zero

Table 4.1 below shows the results of those tests:

<table>
<thead>
<tr>
<th>Perception</th>
<th>Week 1</th>
<th>Week 8</th>
<th>t Stat</th>
<th>P two-tail</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Variance</td>
<td>Mean</td>
<td>Variance</td>
</tr>
<tr>
<td>Difficulty</td>
<td>3.42</td>
<td>0.82</td>
<td>3.83</td>
<td>1.05</td>
</tr>
<tr>
<td>Enjoyment</td>
<td>3.38</td>
<td>1.21</td>
<td>2.94</td>
<td>1.45</td>
</tr>
<tr>
<td>Confidence</td>
<td>3.73</td>
<td>1.2</td>
<td>3.21</td>
<td>1.58</td>
</tr>
</tbody>
</table>

Table 4.1: Paired t-Tests on Change in Perceptions
The P values are all very low (< 0.01), so we have good cause to reject each of the null hypotheses and can therefore conclude that the likelihood is that the changes in mean perceptions are not by chance.

The students themselves gave some insight into why they were not having such a positive learning experience when it came to the programming unit. When queried on their perception of collaborative learning being introduced into the unit, an overwhelming number of students believed that such an environment would not only have a beneficial impact on their learning outcomes, but also make studying programming more engaging, interactive, and fun. See Figure 4.4 below.

![Figure 4.4: Student Perception of Collaborative Learning](image)

Students might not have been aware of the pedagogical values of learning collaboratively, but they believed that a collaborative environment would enhance their enjoyment, their likelihood of engaging, and thus their chance of success in learning to program.

### 4.2.2 Case Studies

In Semester 2, 2007 students from ITB001 and ITB003 were asked to volunteer to be the subject of case studies which involved weekly interviews with the author. Eight students were interviewed each week of semester. The author's involvement in teaching the units used for these experiments afforded the opportunity to have regular contact with students both in and out of the classroom and to build a rapport with students who were then more inclined to openly and honestly discuss their learning experiences.

A small selection of the case studies are summarised below, with pseudonyms used to protect the identity of the students. Some of the more interesting and enlightening dialog is included which reflects each student’s perceptions.
4.2.2.1. Nelly [studying ITB001]

Profile:
Nelly was a domestic female student enrolled full-time in the Bachelor of IT course, majoring in Software Architecture. She was 27, lived in shared accommodation with others of similar age and averaged about 21 hours a week paid work. Nelly was interviewed during her first semester of university while studying her very first programming unit, and had no previous programming experience except “dabbling in HTML”.

Nelly had a history of personal challenges, but in recent years had the determination to fight and overcome a family predisposition for a debilitating health condition.

Pre-ITB001:
Nelly was fairly confident that she would be able to successfully learn to program, and thought she would enjoy the experience. Nelly's expectation was that this programming unit would be neither easy nor difficult. Nelly's perceptions are summarised in Table 4.2 below.

<table>
<thead>
<tr>
<th>Perception of:</th>
<th>Scale of 1 – 5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(1 = low; 5 = high)</td>
</tr>
<tr>
<td>Confidence</td>
<td>4</td>
</tr>
<tr>
<td>Enjoyment</td>
<td>4</td>
</tr>
<tr>
<td>Difficulty</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 4.2: Case Study — Nelly: Pre-Unit Perceptions

Nelly was looking forward to learning to program, but felt at a disadvantage because she didn’t fit the stereotype: no programming experience, and weak maths skills.

She saw programming as something for "nerds", but also added that "nerds are cool" and for that reason, didn’t mind being classified as one if she did well in this unit. Nelly strongly agreed that it makes sense that there are more men than women in programming.

During ITB001:
Nelly spent the expected number of hours on her study of ITB001 as show in Table 4.3.

<table>
<thead>
<tr>
<th>Average hours spent on ITB001 per week</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average hours spent on un/paid work per week</td>
<td>21</td>
</tr>
</tbody>
</table>

Table 4.3: Case Study — Nelly: Study versus Work

Nelly advised that she suffered from anxiety and lack of sleep, and identified this as a distraction to her studies. In weeks where she didn’t dedicate the required 12 hours of study to the unit, it was as a result of lack of time to do so.
Nelly’s confidence during the course of the semester was volatile. She generally welcomed a challenge, and when presented with a difficult topic her approach was to simply try harder, put in more time and keep practicing until it "sunk in".

... Need more practice mastering this topic. Need more exercises. But it all makes sense. ... It gets so confusing. ... Recursion is doing my head in and you get frustrated with it. I think I have it — then do the next exercise and I haven't!

Nelly persisted with hands-on practice even though some topics were "frustratingly challenging". She often reported serious self-doubt in these situations, but was convinced that persistence would eventually pay off.

[I was] stuck on problems for so long. I'm stupid — I'm not getting it. Takes me so long to figure things out. I eventually get it though. I'm a kinaesthetic learner. The [iteration] exercise really got me. I was so pleased when I figured it out.

Nelly rarely sought help from the teaching staff when she was stuck, because she felt like they were “too important to bother”, and felt it would be an "intimidating" experience. She also confessed to keeping up the pretence of being confident and capable, and feared that asking for help would destroy that image and make her appear weak and incapable.

She also expressed disappointment in her friends and peers that they didn’t seem to be putting in enough effort — that they would attend the lectures and workshops, but do little further work themselves. Although preferring to work through problems by herself first, Nelly valued the role of collaboration in learning and ended up mentoring a couple of friends who were struggling with the unit, assuming a leadership role normally undertaken by paid peer mentors.

She always seemed to be up to date with the work and had completed most of the workshop exercises before the workshop, but attended them anyway out of fear of "missing something”. This fear probably resulted from a need to prove her ability (to either herself or someone else) while harbouring some doubt that she could succeed.

Even when Nelly had successfully solved a programming problem, she was not confident in the quality of her solution. She admitted to being a bit of a perfectionist and showed a keen interest in seeing alternative approaches to the same problem.
It’s a case of my usual problem solving strategy of ramming myself into the brick wall of a problem until enough pieces fall off to let it get through. ... But it does the job, so I’m sort of happy. ... It passes all the tests.

Nelly did not initially allocate to a paired workshop, but attended a small number of them due to a timetabling change late in semester. For the purpose of the pair-programming experiment Nelly was considered to be “unpaired”. During those paired workshops that she attended, Nelly happily took on the role of mentor to a number of other students in the class.

Post ITB001:

<table>
<thead>
<tr>
<th>Perception of:</th>
<th>Scale of 1 – 5 (1 = low; 5 = high)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confidence</td>
<td>4 (stable)</td>
</tr>
<tr>
<td>Enjoyment</td>
<td>5 (up from 4)</td>
</tr>
<tr>
<td>Difficulty</td>
<td>4 (up from 3)</td>
</tr>
</tbody>
</table>

Table 4.4: Case Study — Nelly: Post-Unit Perceptions

Nelly enjoyed the unit more than she had predicted she would, even though she found it more difficult than expected. She summarised her motivation and attitude towards studying programming as wanting to do her “absolute best”, and put in a big effort to do so. Nelly's perceptions of programming at the completion of the unit are summarised in Table 4.4.

After completing the exam, but prior to release of grades, Nelly reflected that overall she had been fairly confident of being able to successfully learn how to program during the semester, and predicted a final grade of 7. She said she loved programming and could not wait to do the next programming unit. As Table 4.5 shows, Nelly achieved her predicted grade of 7.

<table>
<thead>
<tr>
<th>Final grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expectation:</td>
</tr>
<tr>
<td>- pre-ITB001</td>
</tr>
<tr>
<td>- pre-exam</td>
</tr>
<tr>
<td>Actual</td>
</tr>
</tbody>
</table>

Table 4.5: Case Study — Nelly: Grade Expectation versus Actual

4.2.2.2. Jane [studying ITB001]

Profile:
Jane was a domestic female student enrolled full-time in the Bachelor of IT course, with (at that stage) unchosen major. She was 20+ and worked part-time, averaging about 5.5 hours
a week. Jane failed her first attempt at this unit in the previous semester, which constituted her first taste of programming. She cited “lack of time” as the main factor for failing, and has since reduced her working hours as a result.

_Pre-ITB001:_

Jane had little confidence from the outset that she would be successful at learning to program, but at least was convinced she now had the time to give it her best shot. Jane figured she would enjoy the unit even though it was going to be hard but expected (and hoped) to pass. Jane’s perceptions of programming are summarised below in Table 4.6.

<table>
<thead>
<tr>
<th>Perception of:</th>
<th>Scale of 1 – 5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(1 = low; 5 = high)</td>
</tr>
<tr>
<td>Confidence</td>
<td>2.5</td>
</tr>
<tr>
<td>Enjoyment</td>
<td>4</td>
</tr>
<tr>
<td>Difficulty</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 4.6: Case Study — Jane: Pre-Unit Perceptions

Jane predicted that this would be the hardest unit for her this semester, but was confident she now had the time necessary to give it a good shot.

Jane had a very nervous disposition, and was constantly fidgeting. She lacked confidence in terms of both her university studies and personal life. A health condition was a constant distraction, which was exacerbated by elevated anxiety levels. Jane advised that she was constantly stressed, sometimes to the extent of having difficulty carrying out everyday tasks.

Jane confessed that her severe lack of confidence had in the past caused her to withhold assignments from submission because she believed them to be too inferior. She was also reluctant to consult with teaching staff about her assignments for the same reason.

Given her previous experience with learning to program, and her anxiety disorder, Jane had little confidence from the beginning that she could successfully learn to program.

_During ITB001:_

Jane reported that she was spending a decent amount of time on her ITB001 studies, as Table 4.7 shows.

<table>
<thead>
<tr>
<th>Average hours spent on ITB001 per week</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average hours spent on un/paid work per week</td>
<td>5.5</td>
</tr>
</tbody>
</table>

Table 4.7: Case Study — Jane: Study versus Work
In the early weeks of the semester, although showing signs of a hectic social life and still coming to grips with her study timetable, Jane was finding the easier concepts a bit of a confidence boost:

*I didn't really learn much last semester. I really like it now that I'm actually doing it. It’s not difficult at all — and that’s a huge relief. I was so worried!*

Jane explained how anxiety affects her learning during workshops:

*It would probably help if I had a look at [the exercise] in my own time. Because of my anxiety I can’t actually understand — think straight at the time. It’s really distressing for me. All I do is smile and nod at the time. I can’t actually take it in.*

There was little clue of Jane’s panic and lack of progress during workshops as outwardly she presented as a vivacious student who rarely asked questions, and when approached, consistently confirmed that she was up to speed and happy with her progress. Jane was a “paired” student for the purpose of the pair-programming experiment but had several different partners throughout the semester due to others’ sporadic attendance at workshops.

Jane had a habit of accepting misinterpreted explanations of concepts without question, and without testing it or proving it to herself. This became evident when she spoke about her understanding of how a non-recursive function call works:

*Like the computer goes through [the function] and unless you write code to stop it, it will just do it again. I didn't realise ... that's how Scheme works. [Give me an example, tell me what you mean.] Like once it’s gone to the bottom of your code it goes back up and starts again at the top. [But only if there is a recursive call]. Oh. Really? [So if your procedure says add 2 plus 3, it will add 2 plus 3 then stop] Really? Well that explains that then! What makes it go around and do it again then? [Recalling the procedure <demo ensued>..]*

As the semester progressed, Jane convinced herself that her struggle was not with implementing code, but with the preparatory problem solving:

*I understand writing procedures more than having to work out the problem. So if the problem was solved logistically, I could probably write the program.*
She continued to have little faith in her ability.

\[ I'd \text{ probably make mistakes even though I thought I got it right. } \]

She reportedly spent a “ridiculous amount of time” on simple assignment tasks, sometimes being in the computer laboratory over night. She approached assignments with extreme caution, once she had actually worked up the courage to start. She was actually “scared” of doing them.

After consultation with counselling services, and on advice from the teaching staff, Jane delayed work on assignment tasks until she had thoroughly revised the workshop material which was designed to develop the skills required for the assignment\(^3\). She was then advised to spend a small amount of time on the assignment task, then seek help if she couldn’t progress further.

Jane was delighted that this approach seemed to work for her by building her confidence:

\[ I \text{ didn't even look at the assignment for 3 hours, I just went through all the other little exercises. Then once I got to the actual assignment, it only took me half an hour. So I thought — maybe I can do it. :-) } \]

Jane continued to struggle through the semester, eventually dropping another unit to ease the workload. She attended two workshops for ITB001 each week, with the intention to use the first as a preview, then taking time to work through the material herself before the second workshop where she would be more confident with the material and less influenced by stress. For this reason, she worked alone in the first workshop each week and with a partner in the second, when she felt she was more confident to interact and contribute to the tasks.

**Post ITB001:**

<table>
<thead>
<tr>
<th>Perception of:</th>
<th>Scale of 1 – 5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(1 = low; 5 = high)</td>
</tr>
<tr>
<td>Confidence</td>
<td>3.5 (up from 2.5)</td>
</tr>
<tr>
<td>Enjoyment</td>
<td>5 (up from 4)</td>
</tr>
<tr>
<td>Difficulty</td>
<td>3.5 (down from 4)</td>
</tr>
</tbody>
</table>

*Table 4.8: Case Study — Jane: Post-Unit Perceptions*

Jane’s perceptions of programming at the completion of the unit are summarised in Table 4.8 above.

\(^3\) This approach is continually recommended to students, but rarely adopted.
Jane said that she had "enjoyed this unit immensely" which seemed incongruous with the degree to which she had suffered with anxiety and lack of confidence and struggled with the learning as a result. Nevertheless, Jane’s expectation of grade increased from 4 to 5 just prior to sitting the final exam and she actually received a grade of 4 (see Table 4.9).

<table>
<thead>
<tr>
<th>Final grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expectation:</td>
</tr>
<tr>
<td>- pre-ITB001</td>
</tr>
<tr>
<td>- pre-exam</td>
</tr>
<tr>
<td>Actual</td>
</tr>
</tbody>
</table>

Table 4.9: Case Study — Jane: Grade Expectation versus Actual

4.2.2.3. **Dave [studying ITB003]**

*Profile:*
Dave was a full-time domestic student and fit the stereotype of a programming student: recent school leaver, highly confident, limited social skills and a keen interest in technology. Unlike the previous two students, Dave was profiled during his second semester of university, while completing his second programming unit. Other previous experience included programming at school, being part of an Internet website development group; building circuitry, as well as a list of half a dozen programming languages that he was familiar with.

Dave lived at home with his family and was not employed in paid or unpaid work.

*Pre-ITB003:*

<table>
<thead>
<tr>
<th>Perception of:</th>
<th>Scale of 1 – 5 (1 = low; 5 = high)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confidence</td>
<td>4</td>
</tr>
<tr>
<td>Enjoyment</td>
<td>5</td>
</tr>
<tr>
<td>Difficulty</td>
<td>2.5</td>
</tr>
</tbody>
</table>

Table 4.10: Case Study — Dave: Pre-Unit Perceptions

Dave expected to enjoy this programming unit, but went to great lengths to convince me that he probably knew it all already. For that reason, he expected it to be fairly easy and was confident of a high overall grade. Dave's perceptions of programming are summarised in Table 4.10.

Dave advised that he had an extraordinary memory and a mind for computing that meant he didn’t need to write much down, nor work through any problem-solving or design process in order to implement an exemplary solution.
I can do a lot of the testing in my mind because I have a mind that can just keep track of variables — millions of variables and just watch code execute in my mind.

Can read and understand code in any language — even if I’ve never used that language before.

**During ITB003:**

Dave dedicated little time to his ITB003 studies each week, as summarised in Table 4.11.

<table>
<thead>
<tr>
<th>Average hours spent on ITB003 per week</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average hours spent on un/paid work per week</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4.11: Case Study — Dave: Study versus Work

Dave attended lectures and workshops with a friend he had gone to school with, but gave the impression that he need not have bothered, given his already significant programming ability. Dave measured his own confidence at the highest level (5) for most of the semester.

[I’m] not bothered to spend more time. It’s just basic at the moment — will work more when it is harder. It just comes natural to me.

Dave’s attitude to the semester-long programming project for assessment was that he would be able to complete it in a couple of weeks and therefore didn’t need to spend the recommended time each of the 13 weeks of semester working on it. He was pretty sure he would be able to “knock most of it over” during the week-long semester break.

The first phase of Dave’s project was graded 30/50 and he was happy with that result. Although his submission was incomplete:

[I] didn’t do the test cases — couldn’t be bothered. I’m lazy — I’ll do the bare minimum ...

He believed the true fault lay with the project specifications supplied to students which were “horrible … badly worded”.

As to the unit as a whole, although his attendance at lectures and workshops was good, Dave seemed to have his own timetable and agenda:

Workshops (all units) go too slow. So I leave it to the end and teach myself.
Dave was a “paired” student for the purpose of the pair-programming experiment, with his partner being his high school buddy. This relationship seemed to motivate him to attend the workshops and he considered himself a mentor for his pair partner who, although having similar experience, seemed to be less confident about his programming skills.

During workshops, Dave would often take part in the activities, or contribute for a short while until he lost interest and until he found something more appealing to do. As a result, some tasks remained unsolved and it was not evident (except from his own insistence) that he was actually capable of completing them successfully.

Post ITB003:

<table>
<thead>
<tr>
<th>Perception of:</th>
<th>Scale of $1 - 5$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(1 = low; 5 = high)</td>
</tr>
<tr>
<td>Confidence</td>
<td>3.5 (down from 4)</td>
</tr>
<tr>
<td>Enjoyment</td>
<td>5 (stable)</td>
</tr>
<tr>
<td>Difficulty</td>
<td>3.5 (up from 2.5)</td>
</tr>
</tbody>
</table>

Table 4.12: Case Study — Dave: Post-Unit Perceptions

Dave’s perception of programming after completion of the unit are summarised in Table 4.12. He reflected on the unit as follows:

Lectures would be useful if I listened — ok for basics.

He insisted that he required very little time to complete programming tasks, including the non-trivial programming project required for this unit. Dave seemed loath to spend time early in the semester on project tasks that he was confident he could complete in a short amount of time just prior to submission date. Combined with the fact that other units had similar assessment demands late in semester, this attitude resulted in insufficient time to complete all aspects of the project successfully. Nevertheless, Dave’s confidence did not waiver and he continued to expect a high grade for this unit. As shown in Table 4.13, contrary to his high expectations, Dave achieved a grade of only 4.

<table>
<thead>
<tr>
<th>Final grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expectation:</td>
</tr>
<tr>
<td>- pre-ITB003</td>
</tr>
<tr>
<td>- pre-exam</td>
</tr>
<tr>
<td>Actual</td>
</tr>
</tbody>
</table>

Table 4.13: Case Study — Dave: Grade Expectation versus Actual
4.2.2.4. **Steve [studying ITB001]**

**Profile:**
Steve was a 24 year old domestic student who had recently failed to complete the unit after a number of previous unsuccessful attempts and withdrawn enrolment. Steve presented as fairly confident about his academic ability, but socially inept.

Steve was frequently witnessed in both lectures and workshops listening to music, playing games and refusing to take part in either class discussions or to converse with teaching staff on a one-on-one basis. Given this attitude, his presence in class confounded both his teachers and fellow students. Steve agreed to be interviewed after completion of the unit, in order to help us understand his attitude towards study and his motivation for studying programming.

Steve’s family had been in Australia for 17 years, and he was in the process of moving out of home.

Steve discussed a less than idyllic schooling where he was very shy, had trouble making friends and received a very poor senior high school result. He felt pressure from his mother to study, get a job and financially support the family. Since school, he had completed a Diploma at TAFE in order to gain entry to university. Steve expressed a keen interest in working in the computer gaming industry, but was disappointed that he hadn’t as yet worked in IT at all. In fact, Steve had never had any kind of job, paid or unpaid.

Steve’s academic history at university reflected a very poor result, with a number of units having been repeated, but none, as yet, completed successfully.

*I think it’s just me being lazy. That’s all. It’s doable.*

Asked how best he learns, Steve responded that his preference was for active hands-on repetitive learning:

*Yeah, doing. Information sort of sneaks out when I’m reading ... If I’m doing it — it will still slip out but I can always repeat it because I’ll know what to do. So I can just keep on repeating that action until I get it.*
ITB001 studies:
As Steve was not interviewed prior to completion of the unit like the previous three case studies, there is no record of his perceived confidence, enjoyment and level of difficulty of the unit. Steve did not attend a paired workshop and was therefore considered to be “unpaired” for the purpose of the pair-programming experiment.

<table>
<thead>
<tr>
<th>Average hours spent on ITB001 per week</th>
<th>&lt; 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average hours spent on un/paid work per week</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4.14: Case Study — Steve: Study versus Work

Steve understood the workload requirements to successfully complete this unit, but as Table 4.14 shows, dedicated very little time to study, if any. He admitted that if he was really serious about it, he would have to spend at least three hours per day on a unit.

On repeating a unit, Steve would convince himself he was way ahead, and lose the motivation to work:

> I felt pretty confident — because actually I was getting ahead of the program. I actually stopped myself. I actually felt lazy. It felt good I was getting ahead but like it was a good excuse to let me do something else.

As to programming, Steve said he would have enjoyed it more if he had known what he was doing:

> If I knew what I was doing, I would like it. But I don’t. And I think if I pay more attention and focus on the work and not get sidetracked a lot, I would like it. I like working things out.

Steve blamed his lack of ability to learn programming on his attitude:

> Like it’s because I’m not paying attention at all. You need to be focused on it fulltime or else you will fall behind — I never really got into the knowledge of it. All the ins and outs of programming. Need to find out about it.

Further discussions about Steve’s study ethic revealed that he was seriously distracted with games, and tended to play World of Warcraft at every possible moment, equating to around 16 hours each and every day:

> [I play it] whenever I’m awake. It’s really addictive.
Steve described the *World of Warcraft* environment as a social place where he interacted with his friends. Because he doesn’t have any real people to “hang out” with, he logs on and hangs out there.

*So if I can just get more real life friends that I can meet face to face I wouldn’t have the need to go on World of Warcraft much.*

Asked about his preference for learning environment, Steve said he preferred studying with a friend, or in a small group:

*I want to interact because I want to get better at socialising and it gets lonely if I’m always by myself. You need to be talking with people anyway. It’s good for your health. Right now I’m trying to do as much of that as possible.*

Steve habitually logged on to the game as soon as he woke up, so it was no wonder his attendance at university and studies suffered. Although he showed a little reluctance to continue this habit, he described a sense of commitment to his virtual gaming friends:

*Like, World of Warcraft — there’s a community. So it’s — I feel like I have an obligation to people in the game.*

After moving house and no longer having internet access:

*I don’t [miss playing it]. ... Relieved actually. Because I felt like I had to log on every time but now I don’t have to, because I can’t. So it’s a relief that I don’t have that obligation.*

Steve admitted that his addiction to *World of Warcraft* had stopped him from being serious with his study. He also claimed he had kept up to date with this unit for the first three weeks of semester at which point he needed to take time off due to sickness, and then fell behind with his studies. He eventually withdrew from the unit.

**Update:**

Steve re-enrolled in ITB001 in Semester 1, 2008 (the following semester) and showed a little improvement in at least his attendance at and involvement in workshops. However, his focus was clearly set on completing the assessment items, and completely ignored the workshop exercises that were designed to introduce problem-solving and programming concepts and build the expertise required for the assignments. His demeanour continued
to be that of an introvert; however it was evident that he was making a small effort to communicate with other students outside class. Steve started asking questions during workshops, but seemed to make little progress in terms of problem-solving skills and ability to program.

Steve regularly attended lectures and workshops as well as extra catch-up sessions that were made available for students falling behind. His assignment submissions indicated some small amount of progress in his understanding of the unit content, although a great deal of help had apparently been sought from many of the teaching staff, and we suspect also from other students. Steve sat the final exam for ITB001, but failed the unit.

Steve again enrolled in ITB001 in Semester 2, 2008 and his attendance at workshops was initially very good. He was less distracted in class, and made a habit of asking many questions, not in group discussions, but privately with a tutor. After spending much time with Steve, it became apparent that one-on-one tuition was not only what he expected, but what he benefited from the most. After gaining a little confidence with his tutor, he verbalised his frustration with the wording of one particular task’s instructions. After this was rectified, for the very first time Steve was able to complete an exercise by himself. He was praised for his efforts and it was quite obvious he was very proud of himself and continued with renewed confidence. Surprisingly, the following week he was called on to present his solution to the class, and did so without hesitation — another first. His solution was neat, well described and of fairly good quality.

His acute shyness and lack of social confidence had initially made it difficult for him to verbalise the difficulties he was having. As a result, he had chosen to struggle on by himself. What he is capable of achieving with a little confidence and encouragement remained to be seen.

Unfortunately, to the best of the author’s knowledge, Steve attended no further workshops after the week he presented his solution to class. Perhaps the prospect of being called on again to address the class was too much to bear. Alternatively, the small progress he made may have pushed his confidence into overdrive, lending him to mistakenly believe he could complete the remainder of the unit without attendance. Steve, once again, failed the unit.

4.2.2.5. Summary of Case Studies

What became evident from the case studies is how different students can be to the first impressions that may form when first meeting. Students don’t always present with a public
image reflective of their true potential and incorrect assumptions can be made about students because of this. For example, we saw Nelly who was shy, easily intimidated and devoid of self-confidence, who was able to achieve an extremely good grade. At another extreme, Dave was over-confident to the point of severely underestimating the time and effort required to complete assessment items, and as a result achieved a much lower grade than he predicted. Students, especially early in their degree, are not necessarily forthcoming with an honest self reflection of their ability, confidence level and motivational factors.

For all these students, the common issues affecting their learning outcome seemed to be confidence and determination. The level of ability may actually be less accurate as a predictor of success for learning to program.

Figure 4.5 below plots the four case study students and their grades against levels of confidence and determination — with the lighter shaded area of the figure representing an estimation of the required level of each attribute for success.

![Figure 4.5: Confidence and Determination](image)

Both Nelly and Jane showed a lot of determination to succeed, even though their idea of "success" was quite different. Nelly would have been satisfied with nothing less than a 7, yet Jane was happy just to pass. They each also suffered from lack of confidence, to varying degrees. Jane’s lack of confidence was debilitating, yet Nelly used her inappropriately low confidence as motivation to try even harder.
Dave was so over-the-top confident, it was often difficult to judge his actual ability — and he was always willing to pass the blame for poor marks on to "things beyond his control". He may well be representative of many young male students: competitive and over-confident yet lazy and unmotivated. Dave’s idea of success was satisfying himself that he could program, rather than demonstrating that to his teachers.

Steve was indeed a sad and repeatedly frustrating case — where a glimmer of hope again proved disappointing. One-on-one tuition and constant hand-holding was not enough to get him over the line. It would be interesting to know if, with a little more confidence and more determination to succeed, whether Steve ever passes any university units.

4.2.3 Weekly Surveys
In Semester 2, 2007, ITB001 students who attended lectures were asked to complete a survey each week which required them to quantify their levels of confidence, difficulty and enjoyment of programming in this unit. They were asked to rate each of these three key areas from 1 (low) to 5 (high). Figure 4.6 below shows the average level of confidence, difficulty and enjoyment for ITB001 students over the 13 weeks of the course.

![Figure 4.6: Student Weekly Perceptions](image)

It is not surprising that there seems to be a negative correlation between difficulty and confidence, especially as most of the students in the cohort were not working collaboratively. Only a small number of students took part in pair-programming during this iteration of the experiment, yet the entire cohort of students were asked to complete these weekly surveys. See Section 4.3.1 for details relating to paired and non-paired students in
that semester. When the students found the content difficult, their confidence levels dropped and they enjoyed the experience less.

A Pearson's correlation analysis of the averages of confidence, enjoyment and difficulty is summarised in Figure 4.7 below.

<table>
<thead>
<tr>
<th></th>
<th>Average confidence</th>
<th>Average enjoyment</th>
<th>Average difficulty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average confidence</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average enjoyment</td>
<td>0.70</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Average difficulty</td>
<td>-0.58</td>
<td>-0.65</td>
<td>1</td>
</tr>
</tbody>
</table>

**Figure 4.7: Correlation Coefficients of Confidence, Enjoyment & Difficulty**

A Pearson Correlation (or correlation coefficient) measures the linear relationships between variables, with an $r$ value of 0 indicating no relationship, and an $r$ of 1 (or -1) indicating a perfect positive (or negative) relationship between the variables.

According to this analysis, there is a strong positive correlation between enjoyment and confidence ($r = +0.70$), and strong negative correlations between difficulty and each of enjoyment ($r = -0.65$) and confidence ($r = -0.58$).

By way of comparison, during subsequent iterations of the pair-programming experiment where the entire cohort of students were exposed to a collaborative learning environment, students themselves reflected that they enjoyed the learning experience, found the support of their pair partner to be invaluable to both confidence and learning. Section 4.3.1.4 contains more detailed feedback from students during the pair-programming experiments.

In terms of students' study habits as shown in Figure 4.8, many students were spending far less than the required amount of time on study for this unit. The red bars in Figure 4.8 after weeks 4, 8 and 12 indicate when items of assessment were due throughout semester.
4.3 Pair-Programming Experiments

In-class pair-programming experiments were conducted over several teaching periods from which data was used to measure the effect of pair-programming on student learning. A detailed description of the introductory programming units taking part in these experiments can be found in Section 3.3.1. For the purpose of these experiments, students were categorised as "paired" if they attended more than half the workshops for the unit (that is, six or more), and actively worked with another student (or two) using the pair-programming protocols.

Subsequent cycles of the pair-programming experiments provided the opportunity to implement changes to the pair selection process, pair-programming protocols and the learning environment based on observations and qualitative data gathered at each iteration in order to further improve the learning and teaching process. These changes manifested in the development of a unique learning environment which best supported the students and built a culture of collaboration and open communication.

Effective pair selection proved to be pivotal in ensuring that students had every chance of getting along with their pair partner and maximising the opportunities for them to efficiently collaborate with each other.

The following sections contain quantitative and qualitative results of the pair-programming observations and experiments. This is followed by explanations of how improvements were
made to the pair selection process and to the pair-programming protocol during the longitudinal study of pairing.

4.3.1 Quantitative Results

4.3.1.1. Participation in Initial Experiments

Table 4.15 details the number of paired and unpaired workshops conducted in each semester of the experiment together with details of the students pairing in those workshops.

<table>
<thead>
<tr>
<th>Period</th>
<th>Unit</th>
<th># paired : total workshops</th>
<th># students in paired workshops who completed</th>
<th># paired students</th>
<th>% paired</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007, 2</td>
<td>ITB001</td>
<td>2 : 4</td>
<td>34</td>
<td>16</td>
<td>47</td>
</tr>
<tr>
<td>2007, 2</td>
<td>ITB003</td>
<td>2 : 6</td>
<td>42</td>
<td>28</td>
<td>67</td>
</tr>
<tr>
<td>2008, 1</td>
<td>ITB001</td>
<td>4 : 14</td>
<td>156</td>
<td>64</td>
<td>41</td>
</tr>
<tr>
<td>2009, 1</td>
<td>INB104</td>
<td>11 : 11</td>
<td>336</td>
<td>235</td>
<td>70</td>
</tr>
<tr>
<td>2009, 2</td>
<td>INB104</td>
<td>8 : 8</td>
<td>190</td>
<td>159</td>
<td>84</td>
</tr>
<tr>
<td>2010, 1</td>
<td>INB104</td>
<td>7 : 7</td>
<td>309</td>
<td>270</td>
<td>87</td>
</tr>
</tbody>
</table>

Table 4.15: Summary of Workshops and Pairings in Experiments

ITB001 (Semester 2, 2007)

Students attending two of the workshops for this unit were involved in the pairing experiment.

Of the 60 ITB001 students attending one or more of either of the two workshops where the pairing experiment was conducted during the semester, 34 completed the unit and 16 (47%) of those students paired for at least half of the workshops.

ITB003 (Semester 2, 2007)

Students attending two of the workshops for this unit were involved in the pairing experiment.

Of the 49 ITB003 students attending one or more of either of the two workshops where the pairing experiment was conducted during the semester, 42 completed the unit and 28 (67%) of those students paired for at least half of the workshops.

The larger percentage of students in this unit continuing to be paired may have been influenced by:
• the author being both the lecturer and sole tutor for ITB003 students at one university campus;

• for the remainder of the ITB003 workshops at another campus, the author being 1 of 3 tutors each workshop and probably having more time to give pair support and encouragement to students in this group; or

• ITB003 being conducted in a laboratory which enabled the students to use pair-programming to implement their solutions, rather than just using those protocols for problem solving and design as they did in ITB001.

**ITB001 (Semester 1, 2008)**

Students attending four of the workshops for this unit were involved in the pairing experiment.

Of the 168 ITB003 students attending one or more of either of the two workshops where the pairing experiment was conducted during the semester, 156 completed the unit and 64 (41%) of those students paired for at least half of the workshops.

**4.3.1.2. Participation in the Replicated Experiments**

After the experiments ran successfully for two semesters, the encouraging results helped influence a more intensely collaborative learning environment for the entire cohort of students completing the introductory programming units in subsequent semesters.

Student data from Semesters 1 and 2, 2009 and Semester 1, 2010 are included below. No comparisons have been made with the initial experiment data because of the significant differences in the teaching unit. Nonetheless, as the opportunity existed for the experiment cycle to continue, there was value in collecting the data, reflecting on the results and making changes to the processes as deemed appropriate in order to better the chances of success for introductory programming students.

**INB004 (Semester 1, 2009)**

All students in this unit attending workshops were involved in pairing as part of the curriculum. Of the 336 students who completed the unit, 235 (70%) paired for at least half of the workshops.
INB004 (Semester 2, 2009)

All students in this unit attending workshops were involved in pairing as part of the curriculum. Of the 190 students who completed the unit 159 (84%) of them paired for at least half of the workshops.

INB004 (Semester 1, 2010)

All students in this unit attending workshops were involved in pairing as part of the curriculum. Of the 309 students who completed the unit 270 (87%) of them paired for at least half of the workshops.

4.3.1.3. Student Results

The failure rate for students in the introductory programming subject who were paired in this experiment fell to just 5%. Figure 4.9 shows the failure rates for first semester units at QUT 2003 to 2008, and the failure rates for INB104 after a course restructure in 2009.

It should be noted that the other three first semester first year units (ITB002, ITB004 and ITB005) were discontinued as first semester units in their current form in 2009 and have therefore been excluded from the graph from that date.

![Grades 1-2 (Fail) Comparisons](image_url)

Figure 4.9: Failure Rates During Experiments

Although it is acknowledged that other factors may have contributed to this improvement including different teaching staff, subject content, programming language and student cohort, paired students performed significantly better than those who were not paired in
the same semester, with exposure to the same subject structure. Figure 4.10 below shows the distribution of grades for ITB001 student during the initial experiment periods, by comparing paired and unpaired students as well as the entire cohort.

![ITB001 Grades Distribution During Experiment](image)

Figure 4.10: Distribution of Grades During Initial Experiment

Figure 4.11 shows the failure rates for paired students compared to those for unpaired students, for each semester of the experiments.

![Grades 1-2 (Fail) Comparisons](image)

Figure 4.11: Failure Rates of Paired and Unpaired Students
A one-way (single factor) analysis of variance (ANOVA) test was carried out on the data gathered from the pair-programming experiments to determine the likelihood of pairing having an effect on student results. Only the results for the two semesters in the initial experimental period were considered, as the replicated studies were undertaken after the introductory programming unit was redesigned.

The null and alternative hypotheses are as follows:

\[ H_0: \] the mean grades for paired and unpaired students have the same value

\[ H_1: \] the mean grades for paired and unpaired students are not equal

The ANOVA results are summarised in Figure 4.12 below.

<table>
<thead>
<tr>
<th>Groups</th>
<th>Count</th>
<th>Sum</th>
<th>Average</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>paired</td>
<td>80</td>
<td>456</td>
<td>5.7</td>
<td>2.921519</td>
</tr>
<tr>
<td>UNpaired</td>
<td>351</td>
<td>1604</td>
<td>4.569801</td>
<td>4.400114</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Source of Variation</th>
<th>SS</th>
<th>df</th>
<th>MS</th>
<th>F</th>
<th>P-value</th>
<th>F crit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Between Groups</td>
<td>83.22044</td>
<td>1</td>
<td>83.22044</td>
<td>20.16081</td>
<td>9.16236E-06</td>
<td>3.863226</td>
</tr>
<tr>
<td>Within Groups</td>
<td>1770.64</td>
<td>429</td>
<td>4.127832</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>1854.06</td>
<td>430</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[Figure 4.12: Analysis of Variance Results]

SS = sum of squares  
df = degrees of freedom  
MS = mean square  
F = F-ratio (difference between the sample means)  
P = probability, or significance of F

The F statistic is the ratio of the two mean square values. If the null hypothesis were true, the F-statistic would be close to 1.0 (Stockburger 1996). Also, because the test statistic (F) is greater than the critical value (F crit), we can reject the null hypothesis (based on a confidence level of 0.05).

The P-value is very small, and less than the significance level (0.01), so we can say there is strong evidence against the null hypothesis. The results imply that the means differ more than would be expected by chance alone, and conclude that pairing had a statistically significant positive effect on student results.
Students exposed to pair-programming and supported by a collaborative learning environment outperformed the control group of students who worked independently throughout semester in the final exam as well as overall subject results. See Figure 4.13 for a comparison of average exam marks for paired and non-paired students.

![Average Exam Marks (Out of 30)](image)

**Figure 4.13: Exam Performance**

4.3.1.4. Threats to Quantitative Analysis

The comparison of grades was between paired and unpaired students. "Paired" students were categorised as those who attended at least half of the workshops during semester and worked with a partner using the pair-programming protocol described in detail elsewhere in this thesis. "Unpaired" students were those who either attended more than half of the workshops and worked by themselves, or did not attend the workshops. Therefore, the paired students may have achieved higher grades simply by virtue of having attended more workshops than those who were unpaired. It is not unreasonable to expect students who attended classes and were more engaged in the unit material to be those students likely to achieve higher grades.

However, in these initial experiments, pair-programming was only carried out in a select number of the workshops offered for the unit. All other workshops had students working individually, without any of the pair-programming resources offered to the paired students. Therefore, there would have been non-paired students in other workshops who were consistent attendees engaged in the material and likely to achieve good results. Students in non-paired workshops added an additional dimension to the control group.
Another possible influence on the success of paired students in the study is that they were tutored by the author who was very much focused on this research and who engaged with the students in an intensely collaborative manner. Data collection during workshops entailed making contact with every student at least once, taking the roll, recording pairing details and other information relevant to their involvement in a pair or otherwise. Each student was asked specifically about the logistics of their collaboration: how they were getting along, how and when they meet outside workshops, how they resolve issues, etc. They were also asked pointed questions about their individual progress and prompted to talk about anything that needed to be clarified or demonstrated. As the semester progressed, students tended to become more relaxed about brainstorming and thinking out loud with their partner and asking what may have otherwise been thought of as "silly questions" of the author. They became less afraid of opening themselves up for ridicule, because the author encouraged lively, non-critical discussion from the class so students could learn from each other's mistakes. It is not unreasonable to assume that students attending these workshops may have been presented with an environment more conducive to learning programming simply because of the enthusiasm and focus of the author, and not solely as a result of any benefits for which pair-programming itself was responsible.

However, there were a number of students who regularly attended workshops with the author and who worked individually rather than in a pair. (The various reasons for students working on their own in a paired workshop were discussed in Section 4.3.3.) These unpaired 22 students were in the ITB001 workshops in Semester 1, 2008 and represent 14% of students in all the paired workshops, and 34% of students attending these workshops regularly. Because these students paired for less than half the workshops (even though they attended the workshops), they were categorised as "non paired" for the purpose of this research and therefore fell into the control group of the experiment. In the first semester of the experiment (Semester 2, 2007), no students worked alone in the paired workshops. See Table 4.15 for details of paired and unpaired students in each iteration of the experiment.

4.3.1.5. **An Indicator of Expected ITB001 Results**

In order to estimate what grades the paired students in the initial experiment periods (Semester 2, 2007 and Semester 1, 2008) might have achieved had they not participated in the pairing experiment, a comparison is made between one of the subjects in the study
Chapter 4: Results

ITB001 and another subject with a comparable level of technical material, ITB004 Database Systems.

ITB004 teaches database design, the concepts and terminology relating to databases, and involves writing data manipulation statements in Structured Query Language (SQL). Only the grades of students who completed these two subjects concurrently during the experiment period (whether they paired or not) were compared. Therefore, it is reasonable to assume that a student's study of each subject was influenced to the same degree by family and social commitments, employment, competing study commitments as well as attitude to and motivation for study.

During the period of this comparison (Semester 2, 2007 and Semester 1, 2008) grades were awarded to students on a 7 point scale as Table 4.16 below shows:

<table>
<thead>
<tr>
<th>Grade</th>
<th>7 (High Distinction)</th>
<th>6 (Distinction)</th>
<th>5 (Credit)</th>
<th>4 (Pass)</th>
<th>3 (Marginal Fail)</th>
<th>2 (Fail)</th>
<th>1 (Low Fail)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Semester 2, 2007 Cutoff</td>
<td>85%</td>
<td>75%</td>
<td>65%</td>
<td>48%</td>
<td>45%</td>
<td>25%</td>
<td>1%</td>
</tr>
<tr>
<td>Semester 1, 2008 Cutoff</td>
<td>85%</td>
<td>75%</td>
<td>65%</td>
<td>50%</td>
<td>47%</td>
<td>25%</td>
<td>1%</td>
</tr>
</tbody>
</table>

Table 4.16: QUT Grade Cut-Offs

Figure 4.14 below shows the proportion of students who achieved the same grade in the programming subject (ITB001) and the database subject (ITB004). This proportion of students is marked as 0 change on the graph.
Chapter 4: Results

The graph also shows those who achieved a lower grade for ITB004 (indicated as a negative change on the graph) and a higher grade for ITB004 (indicated as a positive change in grade on the graph). Students who did not take part in the pairing experiment (the unpaired students) were able to achieve a similar grade in ITB004 as they did in ITB001. Of those 105 unpaired students, 42% achieved the same grade for both. 28.5% performed better in ITB004 and 29.5% performed better in ITB001.

By comparison, a greater proportion of students who took part in the pairing experiment (the paired students) achieved a better grade in ITB001. Although a significant number (52%) attained the same grade for both subjects, more than 38% of students performed better in ITB001 while just under 10% performed better in ITB004.

This comparison of student grades for two similarly technical subjects further supports the notion that learning programming in a collaborative environment involving pair-programming had a positive effect on student results. One might also expect that students who enjoyed the benefits of pair-programming in ITB001 may well have employed those collaborative learning skills in their ITB004 studies and had a positive effect on their grade for ITB004 too. Had the experiment been able to eliminate the effect of students making any use of these collaborative skills in ITB004, the results shown in the comparison of these two subjects may well have been even more dramatic.

4.3.2 Qualitative Results

The logistics of setting up and maintaining pairs of students for the in-class pair-programming experiments are detailed in Section 3.3.1.8. This process was enlightening as it provided information about the students that might otherwise not have been forthcoming to other students or the teaching team. Not every student had a similar range of experiences, motivations, goals and perceptions about programming. Therefore a formula for allocating to a “perfect” partner could not be unilaterally applied with any degree of success. Students were required to share their perceptions of programming, their background, cultural and social preferences and other specific demographic detail. The results analysis detailed below includes discussion about how students initially chose a partner, and how the collaboration helped direct their study habits.

Pairing was not considered voluntary and all students were asked to self-select into pairs. For some students, the prospect of being able to work with a friend throughout semester was something they relished. Others were initially more reluctant to pair because they
either had not formed friendships with anyone in the workshop or they simply preferred to work alone. This latter group of students was asked to further reflect on their perception of programming, their goals for this unit, and their programming experience, and the teaching assistant then helped to pair them with someone of similar skill levels and goals.

In later semesters, workshop numbers were larger, attendance fluctuated and it was difficult trying to keep students paired as inevitably one of the students was away and/or had dropped out. If students actively resisted the pairing — or their partner withdrew from enrolment or deserted them and they expressed a preference to work alone, they were reluctantly allowed to work by themselves.

When paired, most students displayed a strong sense of responsibility for not letting their partner down. If they made a mistake or failed to fulfil a pair commitment, they were often more concerned about the negative effect it had on their partner’s grade than their own.

One student with a severe (hearing and speech) disability, with the assistance of a transcriber, paired with another student without difficulty. This pair seemed to have no problems communicating. The disabled student could lip-read, had limited speech and communicated effectively by typing and checking the transcript where necessary.

“Engaging” in the pair-programming experiment involved the students first selecting, and then establishing a rapport with another student. Where there existed no significant conflict or imbalance in terms of language, work ethic or skills level, successful social engagement between students had a positive follow-through effect on the business end of the programming tasks each week. By virtue of their social interaction, the students established a productive learning environment for each other on their level. The ego-charged stereotypical student was given the opportunity to demonstrate his impressive IT skills for a peer who may understand the same level of technological jargon and appreciate the display of competitive prowess. Alternatively, the non-stereotypical student who may have harboured reservations about their ability was able to develop a non-threatening learning environment by pairing with a peer of similar experience and level of confidence in the course material.

Once relationships were formed within a pair, the students tended to unwittingly maintain a two-way support structure by having a more personal reason to attend the workshop and engage in the material: they each had a sense of obligation to their partner. They were provided with not only an opportunity to discuss the work and contribute to the pair’s
progress but there was also an expectation by their partner to do so. This peer pressure seems to have more of an influence on the motivation of the novice student than any amount of pressure from the teaching staff. The students’ obligation to, and stake in, their partner’s learning experience had at least as high a priority as any sense of obligation to their own learning outcome. Because it is difficult to play a very passive role in a pair (as opposed to a larger group) students seemed to develop a commendable study ethic while paired.

Pairing of students promoted more involvement of each student compared to previous semesters prior to the experiment which had required groups of 3 to 5 students to work on certain tasks. Unlike larger groups, it is difficult for a student to take a passive role in a pair, as it becomes more obvious that every student is not taking an equally active role.

A student was considered to be one who "paired" if they attended six or more of the weekly workshops during semester (i.e., half of the 12 or 13 workshops), and during those workshops actively worked in a pair-programming environment. These students had been given verbal and written instructions on pair-programming during the first workshop they attended for the semester, together with background information to read. Each week these students were reminded of the distinct roles each partner in the pair was to play. They were encouraged at regular intervals to swap those roles. Open communication and interaction between the pair was promoted by the teaching staff.

In ITB001 and INB104 workshops students were expected to make presentations to the class each week, to demonstrate their approach to solving the workshop exercises. Most pairs opted to make a joint presentation of an exercise solution, rather than take turns. These presentations became a valuable class learning experience, with the presenters providing another perspective to solving the programming problem to the rest of the class, who then provided constructive criticism and feedback to the presenters.

Students were initially only allowed to pair-program in class and not for assessment items. Anecdotal evidence and feedback indicated that students would have liked to been permitted to collaborate on assignments and continue using the protocols they had learned in class rather than only for the benefit of workshop exercises in class.

From Semester 1, 2009 INB104 students were required to submit their major assessment items in pairs. They were also required to submit for individual assessment a Reflective Journal that documented their personal learning experience during the semester. A
framework was provided for students to follow including a critical reflection of their learning experience in terms of what interested them, what was most difficult and how, on completion of the unit, they felt about further studies in the area of each particular technology. Students were also asked to reflect on their experiences with pair-programming and to describe how they collaborated, and if and how they believed the collaboration helped them to learn.

Much of the feedback from students suggested that they believed pair-programming had afforded them benefits which are typically found to occur with collaborative work for example:

- time saving in development;
- less errors;
- knowledge transfer between partners;
- development of teamwork and trust; and
- effectiveness of mentor/apprentice roles.

In Semester 1, 2009, the reflections gave valuable information about the effectiveness of pair-programming from a student’s point of view, especially in terms of the process of pair selection:

- collaborative learning ... did not aid in my learning at all. The reason behind this was that my partners were both international students who were still learning to speak English. This made working in a programming pair difficult.
- I tended to take the role as driver most of the time, I think it’s just my nature to want to be in control.
- I do believe collaborative learning assisted in my learning but I did not entirely enjoy the learning because I did not feel entirely comfortable around my programming partner.
- As for working collaboratively I found it to be a pain really. My partner ... wasn’t experienced with programming and admitted that he didn't really enjoy it. From this I believe that I would have been better doing this unit by myself.
- The most difficult part of this unit was being forced to work with a partner. I did not feel as if my partner was as motivated as I was to do well ...
- Because of my extra experience I was creating complex solutions that my partner could not fully comprehend, and I seemed to confuse the situation.
This negative feedback about pair-programming highlighted the importance of selecting the right students to work together and presented an opportunity to refine the process of pair selection which is detailed in Section 4.3.3.

There was also some feedback about the enforcement of role swapping:

- *We didn't follow the specific changing paradigm but did swap over often. This was because at some point my partner would be on a role (sic) and I would help him with his ideas, and the same went for me. Breaking to stop and swap can lead to ideas being lost and confusion.*

It became apparent that perhaps enforcing strict role swapping was counter-productive and, not wanting to thwart productivity, changes were also made in this regard in subsequent semesters as detailed in Section 4.3.4.

Overall, however, students gave enthusiastic feedback about their experience with pair-programming. In the last semester of the experiment, Semester 1, 2010, a sample of reflections from 111 students was analysed and 90% them had something positive to say about pair-programming, much of it along very similar lines to the following responses which have been gathered since Semester 1, 2009. Each response has been categorised by the perceived benefits of pair-programming that students had identified:

**learning / overcoming difficulty:**

- *I feel that doing things with a partner has helped improve my skills in programming.*
- *... being able to work with my partner made me understand the process easier.*
- *The programming in this unit was easier then I expect (sic), but was not without its difficulties, especially in the later projects. I felt the relative ease of this was due to programming in a pair. I believe this assisted greatly in allowing me to complete tasks.*
- *If we were not in teams I don’t think I would have been able to learn and understand the concept of programming as well as I have.*

**industry-required skills:**

- *Working together to solve a common problem has taught us the valuable social, negotiation and communications skills required in the IT industry.*

**another perspective:**
• This method was useful because I was empowered to see how my partner was thinking about solving problems through the driver-observer paradigm.

• If there was one part of this unit that I would have to applaud for working extremely well, it would, without a doubt be the collaborative learning set up. So many aspects of programming are so different to the normal, everyday way of thinking, that one person might instantly see something that another person could stare at for hours and not pick up.

positive image

• INB104 also dispelled myths about programming for me, including the common belief that programming jobs mean that you are working alone and away from other people. I was glad to hear this.

fun and enjoyment

• One of the things that were most interesting about undertaking INB104 was the collaborative learning aspect. Although I admit I was wondering about that, I also admit that it was really fun ...

• I have enjoyed the programming elements within the course, more so than I previously had thought, perhaps because I worked with a partner.

• Collaborative learning was an important part in making me enjoy this unit as it allowed me to discuss my ideas with a partner, as opposed to trying to figure everything out by myself.

pair commitment/ pair pressure / motivation

• Although a lot of hard work went into this course, I am confident in saying that I would not have done as well without the partner I was given for our practical classes. Having someone with more knowledge and practical experience than me meant that learning was easy and pushed me to keep my academic standards high so as not to let him down. Without his support I may never have been able to master the knowledge that I have gained and without his example to follow I would have not had the motivation to keep up my standard of work.

• I believe that collaborative learning was imperative to my success in this subject. When you have someone else relying on you in a group it is a great motivation to do the best you can, because they are relying on you as well.
In my opinion, the QUT partner system is fantastic. This person who you can confer with and share ideas and concepts, really helped to put to rest any self-doubt or concerns. The partner system made me feel more comfortable about thinking boldly and having someone to provide a second set of eyes to your work.

It has to be said that I had a somewhat negative outlook towards programming ...after a few weeks’ introduction and learning the ropes, I found it not too difficult, and I would say that I actually enjoyed coming to (INB)104 classes. Part of that was heavily credited to the collaborative learning approach though. I found it much easier to tackle a challenge if I knew there was support there should I encounter any issues.

There can be a culture of competitiveness amongst programming students which has the potential to intimidate other IT students. Those who were reluctant to come forward in class to ask a question were more likely to admit to having trouble when approached on an individual level by the tutor. Students who worked individually tended to keep their problems to themselves for fear of being ridiculed or feeling inadequate. Teaching staff consultation times have historically been very poorly attended, and several students taking part in the weekly interviews alluded to the fact that they didn’t want to waste the lecturer’s time with their programming questions, or that they didn’t want the lecturer to be aware of their "inadequacies" as a programming student. They preferred to present with a "capable" front, and struggle through the issues by themselves. This issue was clearly resolved in the collaborative environment where students obviously had the explicit empathy of their partners and they developed a partnership of trust and confidence in their ability to solve their problems or to seek help from teaching staff if together any issues remained unresolved.

Further positive feedback came from the unit's PASS leader in Semester 1, 2009 who worked with students on a peer-to-peer level every week, and facilitated discussions about the unit content:

I think the peer 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).
Because the collaborative environment encouraged such an interactive environment, students very quickly became accustomed to talking both within their pair and in front of the class. They also seemed less reluctant to approach other students outside their pair to discuss issues with the exercises. One of the most pleasant side-effects was that students more readily approached and talked to the teaching staff. They seemed to gain confidence from asking questions of each other, realised they weren't considered a fool because of it, and as a result had the courage to ask what they may have previously thought to be "silly" questions of the tutor. This way, they were less likely to remain "stuck" for long when struggling to understand a new concept or debug existing code. In previous semesters, students had not been inclined to seek help with issues they have with programming, but with pair-programming they enjoyed the non-threatening support they received from their peers.

4.3.3 Improvements to Pair Selection

At all stages of the longitudinal study, students have been allowed to self-select their pair partner. However, initially there was a substantial amount of involvement by tutors in pair selection. Suitable pairings were seen to be those where the levels of previous experience and perceived programming aptitude were similar. The process used for pairing students together was detailed in Section 3.3.1.8.

When self-selecting into pairs, very few students refused to work collaboratively. There were always students who voiced an aversion to "group work" and at first glance they may have believed that the type of collaboration required for pair-programming was simply another form of group work. However as they learned to pair-program and follow the protocols, they grew to understand that collaborative learning was not a matter of dividing up an assessment item and being responsible to others for development of a chunk of the solution. It was the consistent adherence to pair-programming protocol during the learning process that students come to understand (at least by the end of semester), that had a positive influence in their success.

It was observed in a couple of cases that students would rather stay in a poorly performing pair than talk honestly with their partner about their performance or work load contribution issues. For some students, this may have been the first time they had been exposed to this type of conflict and perhaps were poorly equipped socially, and lacked the confidence, maturity and communication skills to resolve the situation themselves. This highlighted the need for tutors to continue to play an integral part in the continued support
of pairs throughout semester. The need for keeping attendances and recording pairings each week presented the opportunity to speak with each student about their progress and to observe and assess the relationships between the students as they worked.

Some instruction was given about how to choose an appropriate partner for pair-programming. We had found in the existing literature that some pairings of dissimilar programming skill levels worked well (see (Vanhanen and Lassenius 2007)). Others reported more benefit in matching programmers of only slightly different skills (Van Toll, Lee et al. 2007), while others agreed that matching similar skill levels was more beneficial (Bevan, Werner et al. 2002; Canfora, Cimitile et al. 2003; Williams, Layman et al. 2006). Our own research showed that some students with similar programming experiences work well together as they work at the same pace, and have similar goals and knowledge base. Sometimes students of very different levels of experience work well too — as one is happy to take the mentoring role, and the other that of the novice. In other circumstances, different levels of experience cause conflict and frustration which may not be easily resolved in the learning environment. Compatible motivation levels and learning goals were also to be considered. The point was to empower students to make their own assessment of the traits they themselves considered important. Well informed self-selected pairings seemed to suffer fewer problems and it was thought that this was because students took more responsibility for the choice of partner in the first place.

Each iteration of the experiment implemented a more informative and constructive introduction to pair-programming, especially for pair selection, which helped to engage students in pair-programming earlier.

Positive feedback about successful pair selection ensued:

- *More than just a learning companion, ... gave me a guidepost as to my learning progress, and a peer to review my code. Having another person of equal skill level read over my code was of hugely beneficial.*

In the first iteration of the pair-programming experiment, all students who attended paired workshops worked in pairs. For the other semesters, although pair-programming was not optional, students were not forced to remain in a pair if they strongly resisted or if circumstances made it so difficult to work in pairs that it impacted adversely on either of the students' work. Teaching staff talked to students in these circumstances, reiterating the benefits of pair-programming and suggested methods of overcoming logistical issues.
In the end, the students made informed decisions about whether to remain paired or not and received the support of the teaching staff whatever they decided.

4.3.4 Improvements to Pair-Programming Protocol

Students often started the unit being unsure about what to expect and having a definite aversion to "group work". Talk of working in pairs often led students to believe there was group work involved, which to them meant each taking a slice of a programming project, working on it individually then piecing it together to form a whole. Students seemed well aware that this type of collaboration was prone to parasitic behaviour and uneven workloads. Some early effort went into identifying the differences between group work and pair-programming, where both students worked on the same project at the same time on the same computer — thereby each student having equal ownership of all parts of the submission. Students were formally introduced to the protocols of pair-programming to clearly distinguish it from other group work collaboration. In order to convince students of the benefits of pair-programming they were shown testimonials from students in previous semesters who had written overwhelmingly in favour of paired work. An example being:

_We followed the pair-programming paradigm of swapping the Driver and Observer roles more when working on the last portfolio submission than the first, and found that we did complete the tasks much quicker and with less difficulty. After completing this unit, I would definitely have worked with my partner more during all pair assessment tasks from the very start as I found it was more effective and beneficial to my learning than working individually._

Publishing feedback from students who had just completed the unit seemed to have more effect than just extolling the virtues of pair-programming from a teacher's point of view. Another particularly useful quote was one that highlighted the benefits to even experienced programmers:

_I have also learnt that no one really knows everything, and that you gain nothing by being cocky. I learnt a great deal through and by the assistance of my pair partner who had basically no programming experience at all._

Apart from these testimonials, we also introduced a video which helped explain how working collaboratively can save a great deal of time and stress. The Invisible Gorilla (Chabris and Simons 2010) was used as evidence of "selective attention", which pair-
programming remedies. An instructional video by NCSU (NCSU 2010) introducing pair-programming was also played in the first workshop of semester.

According to the established protocols of pair-programming, it is advisable for roles to be swapped at regular intervals. Initially in the pair-programming experiments, students were asked to swap the roles of Driver and Navigator at intervals of approximately 15 minutes. If workshop exercises were found to be quite simple and completed in a shorter period of time, then that was also considered an appropriate time to swap roles. Teaching staff enforced regular swapping of roles by reminding students throughout workshops.

Feedback from students indicated that strict enforcement of role swapping was often counter-productive. When they were on a "roll" in terms of designing or implementing a solution students found it annoying to have to change places, just because they were told to. They considered it more productive to continue in the same roles while a piece of related code or design was being fashioned successfully. Once they encountered a problem or were unsure of the next step, they were happier to swap roles as they felt it was as useful as taking a break and clearing their heads.

Using one computer, keyboard and mouse is the established protocol for pair-programming which was implemented in the early iterations of the experiments. This ensured that the two students were focused on the same task, and were in the specific roles of Driver and Navigator. This often worked very well. However, when the tasks become more complicated and some amount of research and experimentation was required in order to complete them, it benefitted the students more to have a second computer within reach to use for this purpose. Teaching staff then required a little more vigilance to ensure that students did not end up working on separate tasks, but rather used the second computer for intermittent research work only.

As the majority of the students involved in these experiments were young men, many were reluctant at first to sit too close to each other in class. Their chairs would be a respectable distance apart, which made sharing the same screen very difficult. It became a job of the teaching staff to gently persuade them to move closer together and fashion the screen into a position where it could be easily seen by both students. Students mostly remained in their own chairs for the duration of the workshops, and simply moved the keyboard and mouse (or laptop) to the other person at a role swap. This made it a quicker process and less distracting to themselves and others in the room than swapping chairs.
Workshops during the pair-programming experiments were boisterous and engaging. Students were encouraged to maintain verbal communication with their partner throughout, with the Navigator vocalising ideas, instructions, suggestions and the Driver commentating their actions throughout. Because of this, it was often easy to tell which pairs were sidetracked or distracted and needed some gentle persuasion to get back on task. Demonstration of students' programming solutions during the latter part of each workshop gave them the opportunity to explain and demonstrate their work which often became a learning experience in itself. Students in the audience would often see an alternate solution to that which they had developed themselves which often surprised them because they had thought theirs was the only one possible. The presenters were given a critical evaluation with constructive feedback and discussions often ensued about which approach was more appropriate or efficient, and why.

With discussions about assessment items, although students were already interested in the prospect of completing some of the projects on offer for assessment, they were very keen to see exemplars from previous semesters. After assessing the projects, students particularly enjoyed having their own work singled out for display to the class and often volunteered to run the demonstration, adding remarks about the logistics of developing the solution in collaboration with particular areas of skill and interest for each student.

Although students met formally only once a week at workshops to use pair-programming, they were strongly encouraged to make firm arrangements to meet at other times during each week and to at least be in contact with them several times a week. In the first week of semester during the ice-breaker and introduction sessions, part of the discussions were to include timetabling and agreed meeting times. Each workshop the tutor was responsible for taking the role, recording who was paired with whom, and talking to each of the students in the class. For absent students, they would enquire of their partner as to the reason and if it was unknown, this presented an opportunity for the tutor to enquire a little more about how the pair was getting along, how often they met and how they felt they were coping with the work. Any problems with the pairs often arose during these discussions and steps were then put in place to try to resolve them.

Often, students with some previous background or experience with programming are reluctant (and sometimes quite vehemently opposed) to changing their software development habits. However, working in a pair helped to persuade these students to conform to the required conventions for the sake of their team and provided an
opportunity for developing the skills and attributes required of a software developer in industry.

In Semester 2, 2009 for the paired assessment items, we apportioned part of the marks for demonstrated collaborative effort. Students were orally examined after submission of their collaborative portfolios, in order to ascertain the distribution of the work and the means by which they carried out the collaborative work. They were also observed during the course of the semester in workshops so the tutors who marked the assignments had a fairly good idea anyway as to how each pair was collaborating. The effect of this collaboration mark was further incentive for the students to work collaboratively or at least for the dominant of the two students to mentor the other to get them up to speed for the oral examination. Each student in the pair was awarded the same mark, thereby reducing the appeal of parasitic behaviour and inequitable workloads which would likely result in a poorer grade even if the final projects themselves were of a high standard.

It was not difficult to see that to get the most out of pair-programming, the tutor had to be a "collaborative learning convert", so it was important to educate the educators about their role in the workshops and have them committed to enforcing the pair-programming protocols.

4.4 Summary

The Then & Now Survey and Weekly Surveys were undertaken to determine the social and cultural impacts on learning programming and to gauge student perceptions of collaborative learning.

Before the pair-programming experiments were introduced, students found programming more difficult than they had anticipated, were less confident about being able to learn to program and enjoyed it less than they thought they would. These students believed that a collaborative environment would enhance their enjoyment, their likelihood of engaging, and thus their chance of success in learning to program.

The following semester after introducing the pair-programming experiments, students reflected on their experiences of working in pairs and credited this type of collaboration for their increased confidence in the work and their subsequent ability to learn to program, for being able to overcome difficulties with learning the content and for the learning experience being enjoyable. A selection of these reflections is documented in Section 4.3.2.
For students involved in these experiments, those who pair-programmed achieved higher grades than those who worked individually. Paired students also outperformed non-paired students in exams. Results data from another subject of a similar level of difficulty and technical nature undertaken concurrently by the same students supports this conclusion by showing that the paired students achieved a grade greater than what might have been expected had they not had the support of the pair-programming learning environment.

This longitudinal study allowed us to improve the process of pair selection over the various iterations of the experiment after identifying that the main difficulties that students had were with different skill levels. Other issues with incompatible or unproductive pairs related to conflicting motivation levels and learning goals, as well as timetabling and geographic considerations which restricted students from meeting regularly. Teaching staff took less of a role in making pair choices and concentrated on informing students about the types of attributes in other students that may be important to them as a pair partner including level of experience, motivation, timetabling and logistics of meeting, confidence, age, gender and culture.

More time was dedicated at the beginning of semester for students to get to know each other and to make informed decisions about the most appropriate peer with whom to be paired. Inspiring testimonials from previous semesters' students were published which talked about the benefits of choosing a partner wisely.

It was evident from student feedback that much effort should be put into marketing pair-programming to the new cohort from the very first workshop of semester. Pair-programming was introduced with a clear distinction between it and other group work collaboration. Past students' testimonials were shown which passionately espoused the virtues of pair-programming, as were videos that depicted the phenomenon of selective attention and demonstrated pair-programming in action.

Chapter 5 which follows documents anecdotal observations made during the course of this research and draws conclusions from these results and suggests further work that may be carried out to validate the findings of this study.
Chapter 5. Conclusions and Future Work

We hypothesised that pair-programming would benefit students learning to program by addressing social and cultural issues they face.

Students told us that when they found programming difficult, they lost confidence and enjoyed it less. They also believed that they would be more confident learning to program in a collaborative environment which would be more fun and help them to develop sound programming skills.

Existing literature indicated that the collaborative nature of pair-programming in particular demonstrates to students that software development need not be the competitive, socially isolating activity that they imagined. Learning in a collaborative environment can help address confidence issues and make learning programming a less intimidating, and more enjoyable activity. This research provided further evidence to suggest that collaborative learning addresses many social and cultural issues that student face while learning to program, which were summarised in Table 2.2.

The opportunity to collaborate and program in pairs provided support for many students learning programming, where otherwise they may have been less successful (see Section 4.3 for analysis of pair-programming experiment results). The quantitative results of this research show a statistically significant improvement in student outcomes when students learned programming in a collaborative environment using pair-programming.

In-class pair-programming experiments were run over several teaching semesters in introductory programming units at QUT and a comparison of final results of paired and unpaired students was used to determine the effect of pair-programming on learning outcomes. Over the course of this research, students were surveyed, interviewed and asked to reflect on their perceptions in terms of their confidence, difficulty of learning and enjoyment levels as well as their study habits and attitudes.

The experiments we conducted confirmed that pair-programming had a positive effect on students' learning. Both empirical and anecdotal results of our experiments strongly support our hypothesis.
5.1 Outcomes

The surveys and interviews of introductory programming students showed us that barriers to learning included lack of confidence in programming ability and lack of enjoyment of the learning process when the content was difficult. Pair-programming experiments addressed these issues, with students responding positively to the collaborative learning environment and the grades for students who paired while learning to program being significantly higher than those who worked individually.

In the final semester of the initial experiment, only 5% of the paired students failed the subject, compared to a failure rate of 20% for non-paired students. Failure rates fell further in subsequent semesters of the experiment (see Figure 4.9). Students participating in the experiment who were paired not only achieved better overall results in the subject, but they also performed better in the subject’s final exam (see Figure 4.13).

Contrary to the conclusions of a similar study (McDowell, Werner et al. 2002), these results indicate that the paired students were able to independently apply their knowledge to new problems. However, the results of this study concur with more recent findings that students who pair-programmed were more likely to complete the course successfully (Braught, Eby et al. 2008).

Students reflected, during the pair-programming experiments, that because of the collaboration they were more confident in their programming ability, enjoyed learning to program and were able to complete more difficult programming tasks than they might otherwise have done alone.

Pair-programming benefits students while they are learning to program. The experiments that were conducted over several semesters in this longitudinal study confirm this. Failure rates were reduced to as little as 5% for those students pair-programming while learning to program. Previously the failure rate for introductory programming units at the same university had been consistently high, peaking at 41% in 2006. Students have given very positive feedback about pair-programming in terms of it providing the motivation for effective study habits, providing the incentive to follow protocols and conventions and for providing a valuable source of support and encouragement during the learning process. We believe that working in pairs to learn to program provided motivation for consistent effort and attendance and was crucial for improving confidence in and enjoyment of the process. The increased confidence and enjoyment together with quality study habits
helped to overcome issues of understanding the unit content and struggling with things like program syntax and logic.

Both empirical and anecdotal results of our experiments strongly support our hypothesis that pair-programming has a positive effect on learning outcomes by addressing social and cultural issues that students face while learning to program.

Contributions made by this research include confirmation of the benefits of collaborative learning and pair-programming in particular. This research provided a longitudinal study of pair-programming protocols and looked at students’ perceptions of confidence, difficulty and enjoyment. Appropriate pair-selection was identified as crucial for successful pair-programming and the process of pair-selection in key areas of students’ previous experience, motivation and timetabling considerations was refined. A unique learning environment was developed including modification of established pair-programming protocols in order to maximise students’ focus, while providing the best possible support. Pair-programming protocols were modified over the course of the experiments including a relaxation of the timing of role swapping in order to not disrupt or distract pair progress. Pair-programming roles were enforced, student contributions monitored, and a culture of collaborative engagement was developed in the learning environment.

This learning environment is being replicated in the current offering of the introductory programming unit at QUT, and further refinement and improvement of the pair-programming environment is expected with this and each subsequent iteration of the unit.

5.2 Future Work

The case study and anecdotal observations made as a result of being closely involved in the students' learning experience provoked further thought about making good pedagogical use of the sociological data of the student subjects. It would be interesting to observe students in the process of pair-programming and record their perception changes in confidence, difficulty and enjoyment through one or more semesters of learning to program.

To help us understand better how it is that students learn to program there is scope for capturing on video pair-programming in action to document and further study the collaborative interactions between students while they experiment with programming ideas. Capturing and analysing students verbal and physical/spatial interactions during
knowledge construction may give valuable insight into the learning process, the misconceptions novice programmers develop and the process of resolving these issues.

Another computer science education research question we could ask is whether Neo-Piagetian theory can be applied to novice programmers. This learning theory is based on the premise that students reason at various levels of abstraction not directly related to biological maturity (as in classical Piagetian theory), but as a result of experience and after building expertise in a specific problem domain. Where we may often expect novices to perform tasks which require formal reasoning, it may be more appropriate and effective to direct the learning material at a more concrete level.

For a list of computer science education research papers by the author during the course of this research, see Appendix J.
Appendices

Appendix A  ITB001 Unit Outline — Semester 2, 2007

Rationale

All Information Technology students need an appreciation of the problem-solving skills involved in computer programming. Although not all Information Technology graduates will become programmers, all IT professionals need to understand the challenges and constraints that arise during software development.

This unit provides a hands-on introduction to the skills involved in solving computational problems using computer programs. It uses numerous worked examples to illustrate fundamental programming strategies and will give you practical experience in writing computer programs. Successfully completing the unit will provide you with a sound basis for ongoing development of your programming skills and an appreciation of the technical issues that must be considered when working with programming staff.

Aim

This unit aims to give you a positive introduction to the analytical skills required in computer programming. It assumes you have little or no previous programming experience. The unit emphasises generic programming concepts and related problem-solving strategies. The skills you learn in the unit will be applicable to a wide variety of commonly-used, industrially-significant programming and scripting languages.

Since many different programming and scripting languages are used in industry, the unit aims to present generic problem-solving strategies that are applicable to a wide variety of commonly-used languages. For concrete illustrations and practical exercises, the unit uses the Scheme programming language. As a teaching language Scheme has the advantage of a simple, freely-available, interpretive programming environment which allows students to start developing programs straight away. The problem-solving strategies embodied by Scheme programs can be transferred readily to industrially-significant programming languages such as Java and C#, and scripting languages such as Perl.

(QUT 2010(1))
Appendix B  ITB003 Unit Outline — Semester 2, 2007

**Rationale**

Understanding how and why modern Object-Oriented Programming approaches enable much greater efficiency and flexibility is essential for graduates working in the IT industry. This core unit builds on the knowledge and skill you gained from Problem Solving and Programming (ITB001) or Software Development 1 (ITB111), bringing you to a level of understanding and skill required for all majors within FIT. This unit prepares you for further studies in a range of subject areas, all of which require a solid understanding of object design and implementation, whether you aim to follow a software engineering, data communications or Information Systems path.

Emphasis is given to helping you design, specify and implement software using an industry best practice approach.

**Aim**

This unit aims to develop your professional and technical capabilities, extending the "procedural" programming and related problem-solving skills you have already gained, into an Object-Oriented approach using an “industrial strength” OO programming language. We aim to give you a solid foundation in the principals of OOP, including encapsulation, polymorphism and inheritance, allowing you to solve real-world problems using the Object-Oriented design paradigm; objects, object behaviour, object properties. The unit uses programming tasks framed in a realistic scenario to further develop your familiarity with the software development lifecycle, the constraints and requirements of and the associated communication requirements within the IT industry.

2010(1))
Appendix C  ITB001 Unit Outline — Semester 1, 2008

Rationale

Computer programs (and scripts) are the fundamental way in which we tell computers how to solve new problems. All Information Technology students need an appreciation of the skills involved in programming. Although not all Information Technology graduates will become programmers, all IT professionals need to understand the challenges and constraints that arise during software development.

Developing a program to solve a computational problem involves two steps. Firstly, you must devise an "algorithmic" solution to the problem, i.e., a sequence of well-defined, unambiguous instructions to follow in order to achieve the desired outcome. Secondly, you must "code" your solution in a form that a computer can interpret, using an appropriate programming (or scripting) language. Through numerous worked examples and practical exercises, this unit will give you hands-on practice at both of these skills.

Successfully completing the unit will provide you with a sound basis for ongoing development of your programming skills and an appreciation of the technical issues that must be considered when working with programming staff. No prerequisite knowledge of computer programming is assumed.

Aim

This unit aims to give you a positive introduction to the skills required in solving computational problems and implementing solutions in a programming (or scripting) language. Although some theoretical aspects of computer programming are introduced briefly, the overall emphasis of the unit is programming practice. The unit emphasises generic programming concepts and related problem-solving strategies. The skills you learn in this unit will be applicable to a wide variety of commonly-used, industrially-significant programming and scripting languages.

Since many different programming and scripting languages are used in industry, the unit aims to present generic problem-solving strategies that are applicable to a wide variety of commonly-used languages. For concrete illustrations and practical exercises, the unit uses the Python language. Python has a simple, freely-available, interpretive programming environment which allows students to start developing programs straight away. The problem-solving strategies embodied by Python programs can be transferred readily to industrially-significant programming languages such as Java and C#, and scripting languages such as Perl.

(QUT 2010(1))
### Appendix D  INB104 Unit Outline — Semesters 1, 2009 – 1, 2010

#### Rationale

Today's modern integrated technology is built on IT systems which run in a range of contexts (e.g. mobile computing, robotics, and web-based systems) using a range of technological solutions such as programming and scripting, databases, web development and network programming.

This team-based unit is an integrated introduction to information technology designed to engage, inspire and inform and will demonstrate the important role that technical system design and development plays in achieving robust operation of a large variety of technological solutions. This unit will give you substantial hands-on, practical learning experiences and will motivate you through engagement in the creative, explorative and meaningful development of technological artefacts that operate in real world contexts.

#### Aim

This unit aims to give you the opportunity to construct small IT systems and to expose you to a wide variety of aspects of system development.

This unit will provide you with a variety of engaging experiences in the broad areas of programming, databases, networking and Internet technologies to broaden your skill set and knowledge about Information Technology systems. This will be accomplished by providing you with a learning framework which will be used to allow you to attempt problems related to technical computing issues so that solutions can be developed which may integrate concepts from different technical areas.

2010(1)

(QUT)
Appendix E  Collaborative Learning Information Sheet

COLLABORATIVE LEARNING in pairs – Information Sheet

What is “collaborative learning?”

Collaborative learning is the grouping/pairing of students for the purpose of achieving an academic goal. Students engage in the course material through interaction with each other. Collaborative learning is generally advocated as an exemplary pedagogical practice where grouped students become responsible for one another’s learning as well as their own [1]. Learning in a collaborative environment becomes a social process where students learn by working with others. Students are interactively engaged in the subject material, observing each others’ approaches to problem solving, keeping each other focused on the task, and being encouraged to verbalise issues and decisions along the way.

What are the benefits to students of learning collaboratively? [2-6]

- synergistic nature of brainstorming and sharing of intellectual resources;
- monitoring the problem-solving process by peer interaction is conducive to successful performance;
- positive affect on cognitive growth and skill acquisition and transfer;
- greater interest and sense of belonging;
- helps students apply algorithmic problem-solving techniques;
- deeper learning and greater motivation;
- higher achievement and course success rates;
- developing skills wanted by industry;
- enhanced confidence in the solution and enjoyment of the process.

What types of students work well together?

Difficult question to answer! Some students with similar programming experiences work well together as they work at the same pace, and have similar goals and knowledge base. Sometimes students of very different levels of experience work well too – as one is happy to take the mentoring role, and the other that of the novice. In other circumstances, different levels of experience cause conflict and frustration which may not be easily resolved in the learning environment. In choosing pair partners for INB104, students should consider:

- demographics (age, culture, gender, background)
- logistics (work/family commitments, meeting times etc.)
- their own level of experience, confidence and motivation and that of any potential partner/s.

As pairs are required to work effectively throughout semester, it is advisable to choose your partner carefully!

How is collaborative learning going to be incorporated into this unit?

Students will be GROUPED IN PAIRS for the entire semester

- Pairs will work together during allocated workshops on:
  - the weekly exercises
  - their portfolio

- During the workshops, each student in the pair will take on a different role for each exercise (or in the case of larger exercises and the assignment tasks, the roles will be swapped at regular intervals eg each 15 mins or so):
  - the “driver” will have control of the pen/keyboard: eg drafting an algorithm; inputting the code; debugging and executing the code
  - the “observer” will be responsible for thinking strategically, asking questions, watching for errors, suggesting alternatives, and providing technical input
  - tutors will enforce the separate roles, remind students to swap roles at appropriate times and encourage intensive and continuous interaction between the students in a pair

- Pairs are encouraged to meet/make contact at other times to complete the weekly exercises eg
  - face to face -
    - directly before or after the workshop each week
    - at another time on or off campus that suits both students
• remotely – by arranging a mutually agreeable time to communicate each week via
  • MSN (or other public chat tool)
  • Blackboard chat (using breakout for private pair discussions)
  • telephone/VOIP (eg Skype)

• Each student must complete the On-Line Self-Assessment Exercises by themselves, and submit an
  individual Reflective Journal (assessment number 3). Therefore collaboration for these items of assessment
  is not permitted.

• It is important to talk to your tutor if you find that you and your partner/s are not working well together (for
  whatever reason). Your tutor will endeavour to assign you to a different partner for the remainder of
  the semester.

FAQ

Is this just “group work” where we break the task into smaller bits, then each do our own part and put it
all together?

No. Learning collaboratively in pairs is intended to be interactive – where both students work on the
same part of the task at the same time.

What do other students think about the concept of learning collaboratively?

In a recent survey of programming students at QUT, they identified the following benefits of learning
 colaboratively:

“Others would pick up on my mistakes and vice versa” (83% agreed, 6% unsure, 11% disagreed)

“More likely to succeed in the unit” (69% agreed, 11% unsure, 20% disagreed)

“Develop sound programming skills & keep each other on track” (83% agreed, 9% unsure, 8% disagreed)

References:

### Appendix F  Then & Now Perception Survey (Semester 1, 2007)

#### Learning to Program - Questionnaire

ITB001 students: sem 1, 2007  Wk:…….

**Donna Teague:** QUT Research Student: Pedagogy of Learning to Program  d.teague@qut.edu.au

<table>
<thead>
<tr>
<th>Gender</th>
<th>Male □ Female □</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>16-18 □ 19-25 □ 26-40 □ 41+ □</td>
</tr>
</tbody>
</table>

**For each question, please tick 1 box only:**

1. What is the level of your computer skills?  
   - 1 = none  
   - 2 = average  
   - 3 = excellent

2. Prior to commencing this unit, what was the extent of your programming experience?  
   - 1 = no experience  
   - 2 = some experience

3. Prior to commencing this unit, how confident were you of a successful completion of the unit?  
   - 1 = not at all  
   - 2 = neutral  
   - 3 = very confident

4. How confident are you now?  
   - 1 = not at all  
   - 2 = neutral  
   - 3 = very confident

5. Prior to commencing this unit, how much did you expect to enjoy it?  
   - 1 = not at all  
   - 2 = neutral  
   - 3 = very much

6. How are you enjoying it now?  
   - 1 = not at all  
   - 2 = neutral  
   - 3 = very much

7. Prior to commencing this unit, how difficult did you expect it to be?  
   - 1 = very easy  
   - 2 = neutral  
   - 3 = very difficult

8. How difficult are you finding it now?  
   - 1 = very easy  
   - 2 = neutral  
   - 3 = very difficult

**In what situations do you currently collaborate with other students in relation to fulfilling the requirements of this unit?**

- never  
- work  
- class  
- labs  
- study  
- revision  
- assignments

**What reasons do you have for not collaborating with other students while learning to program?**

- others would pick up on my mistakes and vice versa  
- collaborative programming would be more time-efficient  
- we could help each other develop sound programming skills by keeping each other 'on track'  
- working together provides a sounding board for discussing ideas and developing a plan of attack  
- a socially interactive environment is more fun  
- a socially interactive environment is more conducive to learning  
- I have more confidence learning the content of this unit with support from my peers  
- I am more likely to succeed in this unit if I have interaction with my peers

**Which of the following, if any, make it difficult to work with fellow students?**

- other demands on my time (eg work, family, community commitments)  
- getting to uni (eg distance from home + transport issues)  
- timetabling interferes with social life and/or sleep  
- lack of motivation and/or interest

**Do you give permission for me to contact you for the purpose of following-up your progress during your studies at QUT?**  
Yes □ No □

Name:………………………………………Student No:  ……………………….  
Email:…………………………………………

*Any personal details you supply will be kept confidential and used for the purpose of follow-up contact only. Summary information resulting from this questionnaire may be published in the course of my research, but personal or identifying details will not.*
Appendix G  Interviewee Profile Form

**STUDENT PROFILE**

Name: ____________________________  ID: ____________________________  Part Partner’s Name: ____________________________  ID: ____________________________

Contact #: ____________________________

Previous Programming Exp: ____________________________

Give permission to contact for follow-up after itb001?  □ yes  □ no

Give permission to discuss issues with other itb001 teaching staff?  □ yes  □ no

<table>
<thead>
<tr>
<th>gender</th>
<th>schools (&lt; 20)</th>
<th>male</th>
<th>mature (20+)</th>
<th>schools (&lt; 20)</th>
<th>male</th>
<th>mature (20+)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>full-time</td>
<td>part-time</td>
<td>full-time</td>
<td>part-time</td>
<td>full-time</td>
<td>part-time</td>
</tr>
<tr>
<td>int indicator</td>
<td>dom</td>
<td>int</td>
<td>dom</td>
<td>int</td>
<td>dom</td>
<td>int</td>
</tr>
<tr>
<td>employment (incl hrs)</td>
<td>dom</td>
<td>int</td>
<td>dom</td>
<td>int</td>
<td>dom</td>
<td>int</td>
</tr>
</tbody>
</table>

(int indicator: dom = domestic, int = international student

(employment = paid or voluntary work)

(For ITB003 students) - Reflection on their first programming unit (itb001?)

..........................................................................................................................
Appendix H  Interview Form

<table>
<thead>
<tr>
<th>INTERVIEWS</th>
<th>Workshop: .............................................</th>
<th>Week: ...............</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name:</td>
<td>ID:</td>
<td>Partner’s Name:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>What you’ve learned...</th>
<th>How will you...</th>
<th>How topic builds on your previous understanding?</th>
<th>What you’ve done</th>
<th>Your perceptions:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Important things covered</td>
<td>Remember</td>
<td>Understand</td>
<td>(On a scale of 1 – 5)</td>
<td>Contradicts – Reinforces</td>
</tr>
<tr>
<td>(list topics/concepts)</td>
<td>1 = not at all; 5 = very well</td>
<td>1 = Contradicts; 5 = Reinforces</td>
<td>(Tick if relevant)</td>
<td>(On a scale of 1 – 5)</td>
</tr>
</tbody>
</table>

Comments + observations

Comments + observations

Hrs spent on this unit last week:

Problem (topic/concept stuck on)

Did you get unstuck?

How?

If Paired:

Did collaboration help?

How?

Did collaboration hinder?

How?

General comments:
## Appendix I  Weekly Survey (Semester 2, 2007)

**ITB001**

Indicate which workshop you attend:
- □ Mon 2-4 (A208)
- □ Mon 2-4 (Q216)
- □ Fri 2-4 (s304)
- □ Fri 2-4 (s309)
- □ Fri 6-6 (s304)

<table>
<thead>
<tr>
<th>WEEK</th>
<th>.................</th>
</tr>
</thead>
</table>

### What you’ve learned...

(Identify a topic/concept introduced)

For each question, please tick 1 box only:

- **I** = not at all; 5 = very well

#### How well do you:

- **remember** the topic/concept
  - 1 □ 2 □ 3 □ 4 □ 5 □

- **understand** the topic/concept
  - 1 □ 2 □ 3 □ 4 □ 5 □

#### How does this topic/concept build on your previous understanding?

- 1 □ 2 □ 3 □ 4 □ 5 □

#### What have you done yourself on this topic/concept:

- □ yes  □ no

- **Practised** this topic/concept

#### How confident are you with this topic/concept?

- 1 □ 2 □ 3 □ 4 □ 5 □

#### How much did you enjoy this topic/concept?

- 1 □ 2 □ 3 □ 4 □ 5 □

#### How difficult are you finding this topic/concept?

- 1 □ 2 □ 3 □ 4 □ 5 □

How many hours did you spend on ITB001 last week including the lecture/workshop/prac time?  

---------- hours
Appendix J   Published Papers

The following papers have been accepted and published during the course of this research:


References


Cohoon, J. P. (2007). A good start but we are not there yet. *38th SIGCSE Technical Symposium on Computer Science Education.*


Dierbach, C., B. Taylor, et al. (2005). Experiences with a CS0 Course Targeted for CS1 Success 36th SIGCSE technical symposium on Computer science education St Louis, Missouri, USA, ACM.


Nagappan, N., L. Williams, et al. (2003). Improving the CS1 Experience with Pair Programming. 34th SIGCSE technical symposium on Computer science


Reges, S. (2006). Base to basics in CS1 and CS2. SIGCSE’06, Houston, Texas, USA, ACM.


Williams, L., L. Layman, et al. (2006). Examining the Compatibility of Student Pair-Programmers. AGILE 2006 Conference (AGILE '06), IEEE.


