An Influence Model of the Experience of Learning Programming

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy

Francisca Alejandra Adamopoulos

Master of Computing
Monash University

School of Business IT and Logistics
College of Business
RMIT University

August 2017
Declaration

I certify that except where due acknowledgement has been made, the work is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program; any editorial work, paid or unpaid, carried out by a third party is acknowledged; and, ethics procedures and guidelines have been followed.

Francisca Alejandra Adamopoulos

31st August 2017
Acknowledgements

I would like to express my deepest gratitude and appreciation to all the people who have helped me, encouraged me and prayed for me in this journey. I thank God for bestowing His many blessings and graces in every way and through the people who have supported me.

I thank my supervisors, Dr Martin Dick and Dr Vince Bruno, for their constant support, patience, encouragement and guidance. Thank you for listening to my ramblings and helping me get back on track. To my initial second supervisor, Professor Brian Corbitt, thank you for giving me the opportunity to start this journey. I also thank the Head of School, Professor Caroline Chan PhD, the HDR Coordinator Dr Konrad Peszynski and the HDR admin staff for their support and encouragement.

I thank all my colleagues at RMIT for their constant encouragement and help. I would like to thank Dr Stas Lukaitis (R.I.P) who instilled in me a love and passion for programming and for his constant encouragement and help. I also thank Dr Bill Davey for his guidance and help during my most challenging moments. I also thank all my colleagues who invited me to lunch, spoke to me in corridors and at the kitchen sink to encourage me to keep going. Thanks to Stefan Cronholm and Ulf Melin who ran a qualitative research course on Grounded Theory and for their tips and advice. I also sincerely thank all the participants of this study who gave up their time to do the interviews.

I thank all the people who kindly offered their prayers and encouragement. I thank the people in my local parish community of St Anthony’s, the Christian Meditation group and Sister Helen Duffy RSM for her friendship and encouragement. I thank all the priests who have kindly listened to me, encouraged me and offered their prayers over many years, especially from the Blessed Sacrament Congregation at St Francis and all my local parish priests at St Anthony’s. I thank my spiritual director Sister Jennifer Sanders rsj for her guidance and prayers.

Finally, I express my deepest gratitude and appreciation to my family. I thank my parents and my parents-in-law for helping to look after my children and my home. Eternal thanks to my husband, Arthur, for looking after the children and everything else at home so that I could keep working on this thesis. I also thank Arthur for his help with proofreading and copyediting to make my English more understandable. I thank my children, Michael and Anthony, for their constant love, encouragement and prayers and for reminding me that family is the most important thing.

May the Lord bless you all!
Dedication

I dedicate this thesis to the Holy Trinity,

To Our Blessed Mother Mary,

To all the Saints, Angels and Holy Beings who love us deeply.
# Table of Contents

Declaration ........................................................................................................................................ ii

Acknowledgements .......................................................................................................................... iii

Table of Contents ............................................................................................................................. v

List of Tables .................................................................................................................................... xvii

List of Figures ................................................................................................................................... xix

Abstract........................................................................................................................................... 20

Chapter 1  Introduction .................................................................................................................... 22
  1.1  Motivations of the Research .................................................................................................... 23
  1.2  Research Aims ......................................................................................................................... 25
  1.3  Outline of the Thesis ................................................................................................................ 26

Chapter 2  Literature Review ........................................................................................................... 27
  2.1  Success Factors ....................................................................................................................... 29
    2.1.1  Cognitive Styles and Learning Styles ............................................................................... 29
    2.1.2  Previous Programming Experience ................................................................................. 30
    2.1.3  Previous Studies ............................................................................................................... 30
    2.1.4  Stage in Degree and Degree Major ................................................................................. 31
    2.1.5  Demographics ................................................................................................................... 32
      Gender ...................................................................................................................................... 32
      Age ......................................................................................................................................... 33
    2.1.6  Other Factors .................................................................................................................... 33
    2.1.7  Summary ............................................................................................................................ 33
  2.2  Student Motivations .................................................................................................................. 34
    2.2.1  Motivational Factors ....................................................................................................... 34
    2.2.2  Career Relevance ............................................................................................................. 36
Developing Independent Debugging ................................................................. 164
Relying on Sample Code .................................................................................... 165
Ad-Hoc Problem Solving Behaviours ............................................................... 166
Lack of Planning ................................................................................................. 166
Ad-Hoc Trial and Error ...................................................................................... 167
Unsystematic Approach ..................................................................................... 167
Dependent Debugging ....................................................................................... 168
Coding in Big Chunks ......................................................................................... 169
Learning Task Skipping .................................................................................... 169

5.1.4 Summary .................................................................................................... 170

5.2 Patterns of Collaboration ............................................................................. 172

5.2.1 One Way Dependent ................................................................................ 173

5.2.2 Two Way Co-Dependency ................................................................. 174

Skill Co-Dependency ........................................................................................ 175

Task Co-Dependency / Division of Labour ...................................................... 175

5.2.3 Collaborative Independent .................................................................... 176

5.2.4 Solitary Independent .............................................................................. 177

5.2.5 Variations ................................................................................................ 179

5.2.6 Summary ................................................................................................ 180

5.3 Patterns of Information Use ................................................................. 182

5.3.1 Syntax Verification .................................................................................. 182

5.3.2 Finding the Meaning of Error Messages .......................................... 184

5.3.3 Looking For Sample Answers .......................................................... 186

5.3.4 Code Adaptation .................................................................................... 189

5.3.5 Comparative Analysis ......................................................................... 190
Chapter 5

5.3.6 Beyond the Scope .................................................................................. 191
5.3.7 Summary .............................................................................................. 192
5.4 Chapter Summary .................................................................................... 194

Chapter 6

Chapter Summary ......................................................................................... 195

6.1 Programming Learner Profiles .............................................................. 197

6.1.1 Learner Nature: Perceived Personal Relevance .................................. 200
    Career Goals .......................................................................................... 200
    Interest .................................................................................................. 201
6.1.2 Learner Nature: Learning Trait .......................................................... 202
6.1.3 Learner Nature: Skill Level ................................................................. 202
    Novice Skill Level ............................................................................... 203
    Competent Skill Level ....................................................................... 203
    Advanced Skill Level ........................................................................ 204
6.1.4 Learning Behaviours Overview .......................................................... 204
6.1.5 Summary ............................................................................................ 205

6.2 Seven Learner Profiles .......................................................................... 206

6.2.1 The Reluctant Beginner ...................................................................... 206
6.2.2 The Willing Beginner .......................................................................... 209
6.2.3 The Keen Beginner ............................................................................ 211
6.2.4 The Budding Manager ........................................................................ 214
6.2.5 The Budding Practitioner .................................................................... 216
6.2.6 The Budding Developer ...................................................................... 218
6.2.7 The Advanced Developer .................................................................... 221
6.2.8 Summary ............................................................................................ 224

6.3 Pathways of Student Learning ............................................................... 227
Keen Beginner -> Budding Developer -> Advanced Developer ..................................... 227
Reluctant Beginners & Willing Beginners -> Budding Managers ................................ 228
Willing Beginners -> Keen Beginners -> Budding Managers ...................................... 228
Willing Beginners & Keen Beginners -> Budding Practitioners .................................. 229
No Progression ............................................................................................................. 230
6.4 Relationships ........................................................................................................ 231

6.4.1 Learner Nature Relationships ............................................................................. 231
Perceived Personal Relevance Influence on Core Learning Perspective .................. 232
Perceived Personal Relevance Influence on Patterns of Collaboration ..................... 232
Perceived Personal Relevance Influence on Patterns of Information Use ................. 233
Learning Trait Influence on Learning Behaviours ..................................................... 234
Skill Level Influence on Learning Behaviours ......................................................... 235

6.4.2 Learning Behaviour Relationships ...................................................................... 236
Core Learning Perspective Interrelationships ......................................................... 236
Core Learning Perspective Influences on Patterns of Collaboration ......................... 236
Core Learning Perspective Influences on Patterns of Information Use ..................... 237
Learning Behaviour Influences on Learner Nature ............................................... 238

6.4.3 Profile Preferences for Collaboration ................................................................. 239

6.4.4 Profile Information Behaviour ............................................................................ 241

6.4.5 Summary ........................................................................................................... 243

6.5 Chapter Summary .................................................................................................. 244

Chapter 7 Literature Comparison and Discussion ...................................................... 245

7.1 Student Characteristics – General ........................................................................ 245
Age ............................................................................................................................. 246
Gender ....................................................................................................................... 246
List of Tables

Table 2-1 - Fixed and Growth Mindset Tendencies (Dwek 2008) ................................................. 43
Table 2-2 - Poor Learning Tendencies (Baird & Mitchell 1991; Carbone 2007) .............................. 50
Table 2-3 - Good Learning Tendencies (Baird & Mitchell 1991; Carbone 2007) ............................ 51
Table 2-4 - Programming Student Preferred Information Sources (Blaho et al. 2013) ..................... 68
Table 3-1 - Student Age Groups ................................................................................................... 92
Table 3-2 - Student Stage in Degree ............................................................................................. 92
Table 3-3 - Student Degree of Study ............................................................................................. 93
Table 4-1 - Programming Learner Profiles Summary ..................................................................... 106
Table 4-2 Participants by Age and Profile ....................................................................................... 109
Table 4-3 Participants by Gender and Profile .................................................................................. 110
Table 4-4 Participants by Degree Enrolled and Profile ..................................................................... 111
Table 4-5 Participants by Stage in Degree and Profile ..................................................................... 112
Table 4-6 Participants by Study Load, Working Hours Outside Studies and Profile ......................... 113
Table 4-7 Preconceived Expectations of Participants by Profiles ..................................................... 116
Table 4-8 Participants Previous Studies of Programming by Profile .............................................. 118
Table 4-9 Participants Previous IT Knowledge / Skills by Profile .................................................. 120
Table 4-10 - Mindset of Participants by Profile ................................................................................ 121
Table 4-11 Participants Self-Efficacy for Programming and IT by Profile ....................................... 123
Table 4-12 - External Forum Usage ................................................................................................. 143
Table 4-13 - Internal Forums .......................................................................................................... 144
Table 5-1 - Problem Solving Behaviours ......................................................................................... 160
Table 6-1 - Programming Learner Profiles Summary ....................................................................... 199
Table 6-2 - Profile Preferences for Collaboration ............................................................................ 240
Table 6-3 - Profile Information Behaviour

Table 9-1 Preliminary description of the Willing Beginner Profile

Table 9-2 Preliminary description of the Budding Manager Profile

Table 9-3 Early summary of the Programming Learner Profiles

Table 9-4 Intermediate concept comparison between Use of Online Information Sources and Dependency

Table 9-5 Intermediate concept comparison between Patterns of Information Use and Dependency

Table 9-6 Intermediate concept comparison between Patterns of Information Use and Motivations

Table 9-7 Intermediate concept comparison between Strong Ownership and Positive Behavioural Engagement

Table 9-8 Intermediate concept comparison of Weak Ownership and all Engagement types

Table 9-9 Intermediate concept comparison of Conceptual Difficulties with itself
List of Figures

Figure 2-1 - Organismic Integration Theory (OIT) (Ryan & Deci 2000) .......................................35
Figure 2-2 - Student ownership of learning model (Conley & French 2014) .............................39
Figure 2-3 - Biggs 3P Model of Learning (Biggs & Tang 2007) ..................................................46
Figure 2-4 - Ways students experience programming model (Bruce et al. 2004) .......................53
Figure 2-5 - Challenging Nature of Programming Theory (Dunican 2006) ...........................76
Figure 2-6 - Taxonomy of Novice Computer Programming Learner Types (Dunican 2006) ......77
Figure 3-1 - Research Plan: theory building process (Eisenhardt 1989) combined with Grounded Theory (Strauss & Corbin 1998) (Bruno (2011)) ..................................................85
Figure 3-2 - Transformation of data from raw collected data to theory (Bruno 2011) ...............88
Figure 4-1 - Model of Influences on Learning Programming ...................................................100
Figure 5-1 - Model of Dependency ............................................................................................172
Figure 8-1 - Essence of the Theory of Influences on the Student Learning Experience of Programming - Model of Influences on Learning Programming and Model of Dependency ....273
Figure 9-1 - List of sources (interviews) coded ........................................................................293
Figure 9-2 List of high level nodes (concepts) ...........................................................................294
Figure 9-3 Earlier List of the Learning Concept .........................................................................295
Figure 9-4 Evolved List of the Learning Concept .......................................................................295
Figure 9-5 Earlier list of Sources of Help ...................................................................................296
Figure 9-6 Evolved list of Sources of Help ................................................................................296
Figure 9-7 Earlier list of Social Networks ..................................................................................296
Figure 9-8 Evolved list of Social Networks ................................................................................296
Abstract

Learning to program is difficult for many students all over the world with programming courses often experiencing high failure and attrition rates. The teaching of programming is still considered a major challenge by educators. At the same time, programming is becoming a key skill required not only of IT graduates but also of students in other disciplines and is becoming more important to a wider range of people. Today’s university students also practice their learning in an extended learning environment that extends well beyond the classroom. There has been considerable research into the teaching of programming in the computing education field, with many studies focussing on content and delivery. More recently, researchers have recognised the need for a greater understanding of how students experience learning to program, from the student’s perspective.

This study contributes to this growing body of knowledge by exploring, in depth, the wide range of influences on the student learning experience of programming. A qualitative study was conducted that interviewed 31 Information Systems students about their experiences in learning programming. The interview transcripts were analysed using a Grounded Theory methodology. A new theory of the Influences on the Student Learning Experience of Programming was developed from the analysis, which is more holistic and comprehensive than previous theories.

The learning experience of programming involves a complex interaction of a wide range of influences. A major influence is the student’s Perceived Personal Relevance towards programming. Students who perceive that programming is relevant to their future career goals are far more motivated to learn it. Perceived Personal Relevance, together with Learning Trait and Skill Level describe the Learner Nature of the student, which influences their Learning Behaviours. The influences within Learning Behaviours include Core Learning Perspectives (Ownership of learning, Learning Task Intent and Problem solving Behaviours), Patterns of Collaboration and Patterns of Information Use.

Patterns of Collaboration describe how students interact with and use their Personal Networks, and include four levels of dependency: One Way Dependent, Two Way Co-Dependent, Collaborative Independent and Solitary Independent. Patterns of Information Use describe the different ways students interact with and use their information sources.
The theory includes Programming Learner Profiles, which encapsulate the relationships and influences between Learner Nature and Learning Behaviours. Each profile describes, in essence, the nature and behaviour of different types of students. Seven distinct Programming Learner Profiles were identified in the study: Reluctant Beginner, Willing Beginner, Keen Beginner, Budding Manager, Budding Practitioner, Budding Developer and Advanced Developer.

This new theory gives educators a greater insight into what students are thinking and doing when learning to program and potential strategies that can improve learning outcomes.
Chapter 1  Introduction

This research is an in-depth study of the influences on the learning experience of programming, of a group of information systems and business students. Learning to program is difficult for many students all over the world, even in the field of computer science (Goold & Rimmer 2000; McCracken et al. 2001; Robins, A, Rountree & Rountree 2003). Programming courses often experience high failure and attrition rates (Bennedsen & Caspersen 2007), with failure rates of non-computer science majors sometimes being much higher (Guzdial 2009). Programming is also a topic that seems to elicit extreme emotions in students, with some loving it and others hating it (Chetty & van der Westhuizen 2013). “In spite of more than forty years of experience, teaching programming is still considered a major challenge” (Caspersen & Bennedsen 2007).

There has been considerable research into the teaching of programming in the computing education field, with many studies focussing on content and delivery. A recent survey of publications categorised papers into four main areas: curricula, pedagogy, language choice and tools, with the largest number of papers focussing on teaching tools (Pears et al. 2007).

More recently, researchers have recognised the need for a greater understanding of how students experience learning to program, from the student’s perspective (Bruce et al. 2004; Felder & Brent 2005; Guzdial 2015; Jenkins 2002; Robins, A, Rountree & Rountree 2003). “Although perceptive practicing teachers of programming are able to intuitively scope the perspectives of their students when it comes to learning to program, little formal empirical study has been conducted into this phenomenon” (Bruce et al. 2004). “If computing educators are ever to truly develop a learning environment where all the students learn to program quickly and well, it is vital that an understanding of the difficulties and complexities faced by the students is developed” (Jenkins 2002).

This study contributes to this growing body of knowledge by exploring, in depth, the wide range of influences on the student learning experience of programming. It develops a new theory of the Influences on the Student Learning Experience of Programming, which is more holistic and comprehensive than previous theories. This theory will give educators a greater insight into what students are thinking and doing when learning to program and potential strategies that can improve learning outcomes.

This chapter presents the motivations for this study and then presents the research questions being answered. An outline of the research design is also described and, finally, the structure of this thesis.
1.1 Motivations of the Research

This research was motivated by several themes, primarily related to seeking a deeper understanding of the student programming experience in all its’ richness and complexity.

Programming is not just for programmers

Programming is not only a key skill required of all IT graduates, but is now also required to be studied in high school and even primary school (VCAA 2014). There have also been recent initiatives worldwide to encourage people from all walks of life to learn how to code, such as hourofcode.com (Du, Wimmer & Rada 2016). Learning to program is becoming more important to a wider range of people than just those intending to work directly in the IT industry. Many information tools such as spreadsheets, databases and statistical packages now require an element of programming knowledge to be used effectively. People developing online businesses can also benefit from programming knowledge, as can those who have a great idea for the next great mobile app.

Even within IT related disciplines, not every student studying IT has the goal of becoming a programmer, especially in fields such as Information Systems. Many previous studies into learning programming are from the field of Computer Science (Pears et al. 2007; Robins, A, Rountree & Rountree 2003). While this is not surprising, as programming has traditionally been a core skill required of computer science students, the need for programming education has extended to many other disciplines.

More understanding is therefore needed in relation to how students that do not intend to be programmers learn programming. This has even been recognised by some in the field of Computer Science itself. “We need to construct learning opportunities for who the learner is and wants to be, not for the expert that we computer scientists might want them to be... not everyone wants to be a computer scientist, a computing education researcher, or a computing educator” (Guzdial 2009).

This study sought to understand the learning experience of information systems and business students. While this was a natural extension of the fact that the researchers themselves are based in an Information Systems school, the study of such a group of students provides new insights into the learning experience of non-computer science majors. Such a group is more representative of the wider range of disciplines that programming is now relevant to.
Students learn in an Extended Learning Environment

Today’s university students practice their learning in an Extended Learning Environment that extends well beyond the classroom. As well as attending formal lectures and performing assigned tasks in tutorials, they usually have access to all the course materials electronically any time of the day. They have access to an endless amount of materials online. When they are not in class, they are able to search the Internet for online tutorials and sample code.

They can view and post on discussion forums both provided by their teachers and others that are publically available online. They might do tutorial tasks in class but they do their assignments outside of class time. They meet and collaborate with their peers in class, but the collaboration can continue well beyond the classroom via any number of electronic mediums, such as email, SMS and Facebook.

Many previous studies, however, are generally constrained to student activities in the classroom (Pears et al. 2007; Robins, A, Rountree & Rountree 2003). More recently, some researchers are finding that “the classroom experience is no longer central to students' learning. Many students had abandoned textbooks and other teacher-provided resources and are heavily reliant on the Internet as source of help and of learning” (Sheard et al. 2013).

This study sought to explore all of the environments that students learn in, both inside and beyond the classroom. It sought to find out how students search for and use information in all of their learning activities and how they interact with others in all stages of their learning activities.

Holistic understanding of the student learning experience

This study sought to explore, in depth, the learning experience of a wider range of students in all stages of their learning. Many previous studies in programming education focus on a single year level (especially introductory programming students) and often within a small number of courses/units (Sheard et al. 2009). By necessity, studies tend to focus on one specific aspect of the learning experience. While each of these studies is valuable, it can be difficult for an educator to gain a clear overall picture of the wide range of influences at play.
This study sought to interview students from all years of their degree, including graduates. It sought to investigate all the activities students perform in their learning, both inside and outside the classroom. It sought to investigate how students use the Internet, how they use their information sources and how they interact with other students and staff during their learning experience. It sought to understand the learning experience from the perspective of the student, in a holistic way, putting the learner squarely as the focus of the learning (Guzdial 2015). It sought to understand “what the student does” (Biggs & Tang 2011).

As a result, this study has developed a new theory which gives a more holistic understanding of the programming learning experience of students.

1.2 Research Aims

This study sought to answer the following research question:

**What are the key influences on the learning experience of programming students?**

Related subsidiary questions were:

- What influences motivate students to learn how to program?

- What types of learners do we find when teaching programming?

- How do students use their personal networks in learning programming?

- How do students use information sources in learning programming?

This research sought a deeper understanding of what influences the student experience in learning programming. A qualitative study was conducted that interviewed 31 students about their experiences in learning programming, that included their expectations, attitudes, perceptions and learning activities. The interview transcripts were analysed using a Grounded Theory methodology.

A new theory of the student programming experience has emerged from this analysis. The theory describes the wide range of influences on the learning experience and their inter-relationships and provides a more holistic understanding of these influences than previous theories.
1.3 Outline of the Thesis

This thesis is structured as follows:

- **Chapter 2: Literature Review**
  Presents a review of the literature relevant to this study, focusing on the programming learning experience, with the student perspective as the central theme.

- **Chapter 3: Research Design and Methodology**
  Discusses in detail the research questions, research approach and methodology used in the study, including the reasons and justifications for the methods chosen. It describes the research process that was followed and how the theory was developed from the data.

- **Chapter 4: Theory of Influences on the Student Learning Experience**
  The theory that was developed by this study is presented over chapters 4, 5 and 6. Chapter 4 introduces the theory and describes the foundational categories: the students themselves, their Personal Networks and the Information Sources they use.

- **Chapter 5: Student Learning Behaviours**
  Continues exploring the theory, detailing the Core Learning Perspective, Patterns of Collaboration (which includes the Dependency Model) and Patterns of Information Use.

- **Chapter 6: Programming Learner Profiles**
  Presents the core part of the theory, the Programming Learner Profiles. This includes the concepts that define the profiles: Perceived Personal Relevance, Learning Trait and Skill Level. It also presents the seven Learner Profiles that were developed.

- **Chapter 7: Literature Comparison and Discussion**
  Enfolds the theory with the literature, comparing and contrasting it with existing theories.

- **Chapter 8: Conclusions and Future Research**
  Describes how the theory developed answers the research questions and highlights the contributions made by this study, as well as limitations and potential future research.
Chapter 2  Literature Review

This chapter presents a review of the literature relevant to this study. As described in the introduction, there has been considerable research in the field of programming education over the past 40 years. A large part of this previous research has focussed on the areas of curricula, pedagogy, language choice and tools (Pears et al. 2007).

More recent studies have explored the student experience of learning to program. This study sought to understand, in depth, the wide range of influences on the student learning experience of programming. As such, the literature presented in this chapter has the student perspective as the central theme.

This chapter is structured as follows:

- Success Factors
- Student Motivations
- Learning Behaviours
- Problem Solving
- Social Networks and Information Behaviour
- Learner Categorisations

Although researchers have sought to identify one or more factors that could be used to predict success, no single factor has yet been identified that can reliably predict which students will succeed in learning to program. These identified success factors do, however, provide valuable insights into the potential influences on student motivation, but the learning experience is complex and needs to be studied in more depth.

What does motivate students to learn (or not learn) programming has only recently received attention. Motivational factors previously identified are wide ranging but are generally intrinsic or extrinsic. They can include career relevance, dependency on others, self-efficacy and the mindset students approach their learning with.
How students go about their learning has been widely researched. Deep and surface learning is a key perspective but other perspectives have included good and poor learning tendencies and student engagement. Most of these have been general education studies and not specifically about learning programming. Recent studies into how students experience the learning of programming and learning programming outside the classroom have provided valuable insights, but more understanding of learning behaviours specific to programming is needed.

The difficulties and obstacles programming students face in problem solving has also been widely studied. Problem solving activities studied have included the use of strategies, planning, comprehending code, tracking code, writing code and debugging. The majority of these studies have focussed on novice programming students in the classroom. More needs to be known about how students solve their problems outside the classroom.

How programming students interact with others has also been studied, mostly within the classroom or within online communities provided as part of the extended learning environment. More needs to be understood of the social and personal networks that students interact with and how they use these networks in their programming learning experience, both inside and outside the classroom.

There have also been studies into how programming students seek information and what information sources they make use of. Most of these have also been within the formal learning environment, although the use of public programming discussion forums is an exception. More understanding is needed of how students use the wide range of information sources available to them as part of their programming learning experience.

To better understand the wide variation in the way students learn, various studies have created student categorisations to portray the different types of students. Categorisations can provide a way for practitioners to understand their students. These previous categorisations are largely based on novice students and primarily of experiences within the classroom. A more holistic categorisation of the programming student learning experience is needed, so teachers can have a better understanding of the potential types of students learning programming.

All these previous studies provide valuable insights into the student learning experience, but each individually sheds light on a small piece of the puzzle. A cohesive model is needed that incorporates the wide range of influences on the programming learning experience and how they influence each other.
2.1 Success Factors

There has been considerable research in Computer Science education looking at factors that impact on students’ performance in introductory programming courses. Researchers have sought to identify one or more attributes that could be used to predict and/or influence student success in learning to program. These have included cognitive and learning styles, previous programming experience, previous studies (such as maths scores), stage in degree, age, gender and a wide range of other factors.

2.1.1 Cognitive Styles and Learning Styles

Cognitive styles and personality is defined as the “individual differences in organizing information, and processing both information and experience” (Bishop-Clark 1995). Cognitive styles and personality variables have been studied as factors that may correlate with programming success.

Field independence and high reflectivity has been shown to be positively related to programming achievement (Bishop-Clark 1995). A field independent person is better at identifying and isolating relevant information from a complex situation, imposing their own structure/organization to the material. Field independence should also be encouraged through teaching (Mancy & Reid 2004).

Studies that have looked at the learning styles/ cognitive profile groups described by Krause (2000) found that Intuitive Thinkers have tended to perform better in programming than Sensor Feelers (Prasad & Fielden 2002; Woszczynski et al. 2004). Intuitive learners use visual memory cues to trigger retention of information. Thinkers like to have concrete evidence or information to make decisions.

Reflective and verbal learning styles outperformed the active visual ones when correlated with programming performance (Allert 2004). “Students who perform well in programming use more meta-cognitive and resource management strategies than lower performing students” (Bergin, Reilly & Traynor 2005).

A number of literature reviews looking at cognitive styles and personality traits have found that while studies do show correlations between factors and performance, they are inconsistent in explaining individual differences and offer little predictive power (Bishop-Clark 1995; Robins, A 2010; Ventura 2005).

As the different programming phases involve different skills, Bishop-Clark (1995) suggested relating cognitive styles to each distinct phase of programming.
2.1.2 Previous Programming Experience

Various studies have found that students with previous programming experience tend to perform better in programming courses than those students with no previous programming knowledge (Cantwell-Wilson & Shrock 2001; Chinn et al. 2010; Gomes & Mendes 2008; Hagan & Markham 2000; Holden & Weeden 2003, 2004, 2006; Kori et al. 2016). The more languages they already know, the better they perform (Hagan & Markham 2000), but the benefit tends to be more on the first programming course than subsequent courses (Holden & Weeden 2004).

A number of studies have found that previous programming experience influences student’s self-efficacy which, in turn, positively influences performance (Özmen & Altun 2014; Wiedenbeck 2005; Wiedenbeck, LaBelle & Kain 2004). In particular, Wiedenbeck (2005) suggests that it is the knowledge organization that directly influences success and strengthens post self-efficacy.

Other studies, however, have found little or no correlation between previous programming experience and student grades (Allert 2004; Bergin & Reilly 2006; Bergin, Reilly & Traynor 2005; Murphy et al. 2006; Pillay & Jugoo 2005; Ventura 2005; Ventura & Ramamurthy 2004).

On the whole, previous studies have shown that students with previous programming experience will tend to perform better in learning programming than students with no previous experience.

2.1.3 Previous Studies

A range of previous studies have explored the relationship between grades from high school, or other previous studies, and student performance in computing degrees, and programming courses in particular. The majority of the focus has been student’s previous grades in maths and science, with mixed findings.

A number of studies have found a positive correlation between student’s mathematical aptitude and high school scores in maths/science with their performance in programming courses (Altun & Mazman 2015; Bergin & Reilly 2005b, 2006; Byrne & Lyons 2001; Cantwell-Wilson & Shrock 2001). Other studies found high school maths grades to be a clear moderate positive indicator of success in first year programming courses (Bennedsen & Caspersen 2005; Gomes & Mendes 2008; Qahmash, Joy & Boddison 2015).
Other studies, however, have found that previous grades have little bearing on success in computer science and programming (Boyle, Carter & Clark 2002). Ventura (2005) found SAT scores offered little predictive value and Alexander et al. (2003) reported that while good grades may predict good performance in mathematics units, they found “nothing in entry qualifications to indicate which students will be successful in the study of programming”.

2.1.4 Stage in Degree and Degree Major

The majority of studies on learning programming have tended to focus on the first programming courses only (Beise et al. 2003). The studies that have looked at two or more programming courses in the programming sequence suggest that a better understanding of the basic concepts of sequence, iteration and decision occurs during or by the end of the second course (Gomes & Mendes 2008; Sheard et al. 2009; Tew, McCracken & Guzdial 2005). The more advanced concepts can then be learnt more easily once students have mastered how to problem solve and have an understanding of basic concepts (Holden & Weeden 2004; Robins, A 2010).

No correlation has been found between the year level of students and the results of introductory programming courses (Bennedsen & Caspersen 2005; Rountree, N, Rountree & Robins 2002). The situation is complicated by non-computing major students doing programming in later years or treating a programming course as a “filler” (Rountree, N, Rountree & Robins 2002).

Various studies have also looked at degree majors and minors as a potential predictive factor of success. The intended major of Computer Science students has not been found to be a significant factor in programming learning outcomes (Bennedsen & Caspersen 2005; Rountree, N, Rountree & Robins 2002).

Computer Science majors have been found to have a higher probability of passing the first programming course than Information Systems majors (Beise et al. 2003). A number of studies have reported that efforts in tailoring programming courses for non-CS majors, such as Information Systems students, have led to higher pass rates (Forte & Guzdial 2005; Yadin 2014; Yadin & Or-Bach 2008).
2.1.5 Demographics

The influence of gender on programming outcomes has had considerable research interest, and to a lesser extent, age.

**Gender**

The issue of gender imbalance in the field of computing degrees due to the steady decline of female graduates in this field, particularly in the USA, UK and other western countries, has been noted by many studies (Sinclair & Kalvala 2015).

Females, however, have been widely found to perform just as well as males, with many studies finding no correlation between gender and programming learning outcomes (Beise et al. 2003; Bennedsen & Caspersen 2005; Byrne & Lyons 2001; Chinn et al. 2010; Pillay & Jugoo 2005; Rountree, N, Rountree & Robins 2002; Ventura & Ramamurthy 2004). Although females may start their computing degrees with less previous programming experience than their male counterparts, by the end of their introductory courses they catch up, achieving similar levels of mastery of programming concepts as the males (Murphy et al. 2006).

Gender differences have, however, been found in student perceptions and attitudes towards learning programming. Male students have been shown to perceive programming as being easier and have a higher intention to program in the future, compared to female students (Rubio et al. 2015).

Female students have been found to be less confident with their skills (Carter, J & Jenkins 2001) and less comfortable than male students asking questions in class and interacting with their instructor (Alvarado, Cao & Minnes 2017; Sinclair & Kalvala 2015). These differences were even found to be consistent or increase across course levels (Alvarado, Cao & Minnes 2017). Female students have also been found to revise their self-efficacy beliefs earlier than males, which suggests early failures could cause female students to disengage from their studies (Lishinski et al. 2016).

Several studies have also found that female students tend to work more with others and study more with their peers, compared to male students (Chinn et al. 2010; Sinclair & Kalvala 2015). Pair programming was found to be beneficial to help female students build their self-confidence towards learning programming (Carter, J et al. 2011).
Age

Relatively few studies have looked at student age as a predictor of programming success (Rountree, N, Rountree & Robins 2002). Those that have studied it have looked at age in combination with other factors and have found that it was not a good predictor of success in programming learning outcomes (Beise et al. 2003; Bergin & Reilly 2005b; Rountree, N et al. 2004; Woszczynski, Haddad & Zgambo 2005a, 2005b).

2.1.6 Other Factors

Various studies have looked at a wide range of other factors as potential predictors of success in learning programming. Some of these have included:

- **Involvement in Computer and Video Gaming** was found to be a negative influence on success (Allert 2004; Cantwell-Wilson & Shrock 2001).
- **Attribution to Luck** was also a negative influence (Cantwell-Wilson & Shrock 2001).
- **Resource Management Strategies** are a positive influence (Bergin & Reilly 2005b)
- **The Grade the Student Expected to Achieve** at the beginning of the course was the strongest single factor indicative of success (Rountree, N, Rountree & Robins 2002).
- **The Student’s Perception of their Understanding** of the module was strongly correlated with performance (Bergin & Reilly 2005b).
- **Ability to Articulate an Algorithm** in depth and map-drawing style (Simon et al. 2006).
- **Abstraction Ability** had conflicting findings (Sheard et al. 2009). Bennedssen and Caspersen (2008) found no correlation while Murphy et al. (2005) found that better performing students are more able to express abstract concepts. Students have also been found to think more abstractly as they progress through their programming course (Perrenet & Kaasenbrood 2006).
- **Programming behaviour based on Event Pairing** from compilation log data during lab classes showed strong correlation with programming performance (Watson, Li & Godwin 2014).

2.1.7 Summary

Despite considerable research into success factors over many years, no single factor has been found to provide a clear determinant of success. The experience students go through in learning programming is complex and needs to be studied in depth. The real picture is best summarised by Sheard et al. (2009): “given enough time, clear direction, and instruction, students will succeed in programming”.

33
2.2 Student Motivations

While student motivations have been extensively researched in general education, the topic has only been recently investigated in the context of learning programming. Motivational factors are generally intrinsic or extrinsic. How relevant students see what they are learning to their future careers can influence their efforts. Some students are independent learners while others want to be taught. Self-efficacy can influence the learning effort. Students with growth mindsets are more likely to learn than those with fixed mindsets. How students react emotionally to their learning can also be an influence.

2.2.1 Motivational Factors

Motivation is a student’s “willingness, need, desire and compulsion to participate in and be successful in the learning process” (Bomia et al. 1997). Motivational factors have been categorised into two broad groups: intrinsic and extrinsic.

Intrinsic motivation, also known as self-motivation, is the deep desire for learning of a subject area. It originates from within the person and is influenced by one’s self-esteem, self-satisfaction, personal values, needs and drives (Bomia et al. 1997; Jenkins 2001). It leads to a deeper learning approach, greater conceptual understanding and produces learning outcomes which are flexible and transferable (Entwistle 1998; Vansteenkiste, Lens & Deci 2006).

With extrinsic motivation, behaviours are performed to satisfy an external demand or obtain an externally imposed reward. They may be done for their instrumental value and not merely for the enjoyment of the activity itself or for its intrinsic interest. Ryan and Deci (2000) describe a continuum of extrinsic motivations, where the motivation varies depending on the degree to which it is autonomous or self-determined (see Figure 2-1). Five types of extrinsic motivation are described:

- **Amotivation (unwillingness):** lack of an intention to act which may be from not valuing the activity, not feeling competent to do it or not believing it will yield a desired outcome.
- **External regulation (passive compliance):** behaviours are performed to satisfy an external demand or obtain an externally imposed reward.
- **Introjection:** behaviours are performed under pressure to avoid guilt or improve self-esteem
- **Identification:** the person has identified with the value of performing an activity and therefore has accepted its regulation as his or her own.
- **Integrated regulation:** the reasons for performing an action are further internalized and integrated to the self and become self-determined.
The stronger forms of extrinsic motivation involve a degree of autonomy with a feeling of choice and a sense of conviction, where subjects display an “attitude of willingness”. Weaker forms involve a mere compliance or external control where subjects are “externally propelled into action”. These weaker forms lead to a surface approach to learning and fear of failure and produce learning outcomes which are inflexible and not readily transferable (Entwistle 1998). Perceived competence, understanding and the belief of possessing the skills to succeed, helps to internalize an extrinsic goal (Ryan & Deci 2000).

Other forms of extrinsic motivation include social motivation, which is based on the idea that the primary motivator is the desire to please some third party whose opinion is valued (Jenkins 2001). To some extent, social motivation might include fear of failure as a motivator (Biggs & Tang 2007).

Achievement motivation is based on the idea of doing well for personal satisfaction (Entwistle 1998). The student will adopt whatever strategy they believe will allow them to obtain the best results or highest marks (Jenkins 2001). Entwistle (1998) describes achievement motivation as being:

- Competitive, with the possibility of becoming egotistical and selfish.
- Dependant on time-management and organised studying.
- Treating tasks as personal challenges.
Within the context of learning programming, several studies have found that for most programming students, achievement and extrinsic motivations were the dominant influences on their attitudes towards their studies (Jenkins 2001; Kori et al. 2016). Intrinsically motivated students, however, have been found to obtain higher programming results (Bergin & Reilly 2005a). Furthermore, students who have high levels of intrinsic motivation and task value use more metacognitive and resource management strategies (Bergin & Reilly 2005b; Bergin, Reilly & Traynor 2005; Sheard et al. 2009). Attempts at using extrinsic motivations in teaching, such as rewards and student comparisons, seemed to have little impact on performance (Bergin & Reilly 2005a).

Several studies have looked at influences on motivation. Lack of skills when learning programming can cause motivation to change from intrinsic to extrinsic while the presence of skills can increase motivation (Carbone 2007; Carbone et al. 2009). Multimedia arts students doing programming did not have a negative attitude to programming but did expect it to be difficult (Bennedsen 2003).

A number of studies have looked at using different technologies to motivate students to learn introductory programming. The use of digital badges in a Moodle-based introductory programming learning environment was found to be positive, although motivation levels varied among learners and over time (Facey-Shaw et al. 2015). Another study explored the use of mobile technology and social media as tools to support student interactions among themselves in their learning of programming (Maleko, Hamilton & D’Souza 2012; Maleko et al. 2014). Other studies have looked at using gamification-based learning platforms to motivate the learning of programming (Burguillo 2010; Fotaris et al. 2016; Swacha & Baszuro 2013).

### 2.2.2 Career Relevance

General educational research has shown that “when students see the relevance of what they are learning to solve problems and achieve tasks in the real world, especially in desirable occupations, students are more motivated to learn the material and are more engaged in school” (Woolley et al. 2013). Relevance as a teaching strategy was found to promote a higher engagement and academic achievement (Keller 1987; Means, Jonassen & Dwyer 1997). Studies on expectancy-value theories of motivation have found that the students’ perception of the value of the lesson content is a key motivational factor because it influences the student use of cognitive and organizational strategies (Means, Jonassen & Dwyer 1997; Pintrich 1999).
Learning interest orientations can be extrinsic such as students aiming to obtain a qualification which would lead to a job or intrinsic such as students seeking knowledge which would equip them well for their future careers (Beaty, Gibbs & Morgan 1997; Jenkins 2001). Even when students recognise what they are learning is relevant to their long term future careers, however, they still may only be motivated to study content that is perceived to be immediately relevant (Lucas 2001). Career motivation and personal interest can co-exist and mutually reinforce each other (Kember 2016).

How programming students see programming as relevant to their future careers may influence their motivation to learn programming. This topic, however, has not been extensively explored. Programming has traditionally been taught in Computer Science, where students are usually attracted to programming and intend to work in the IT field (Kinnunen et al. 2016).

A number of studies, however, have found that not all programming students necessarily want to become programmers. Curzon and Rix (1998) found that even students taking a second or third programming module viewed programming as “useful career-wise but only as a secondary skill”. Another study found that some first year CS students described their future career plans as the desire to work in their own company, in small start-ups or as managers, rather than in a technical role (Kinnunen et al. 2016).

In a study of engineering students, some students were categorised as “conversational programmers”, who did not necessarily want to be professional programmers but wanted to learn programming so that they could “speak the language and improve their perceived job marketability in the software industry” (Chilana et al. 2015). Even non CS majors who enjoyed learning programming were found to dismiss it as not being applicable to their goals (Hewner & Guzdial 2008).

In a study of what motivates IT students to persist with their degree program, three factors were identified: previous personal contact with IT, the reputation of the IT field and development (self-development and IT skills development) (Korii et al. 2016).

Various attempts have been made to change programming content and/or use technologies to make learning of programming more relevant to students. Games and web-programming were used to illustrate how programming is a relevant skill that is used to solve real-world problems that the students perceive important to have for their future careers (Feldgen & Clua 2004).
A number of studies have attempted to teach programming with LEGO Mindstorms, with mixed results. McWhorter and O’Connor (2009) found that students were more motivated, but mostly for the content rather than relevance. Lykke et al. (2014) also found students more motivated and engaged, but they experienced frustration due to limitations of the robots.

With programming now being taught to a much wider range of students, a deeper understanding is needed of how these students perceive programming as relevant to their careers and how that influences their motivations.

2.2.3 Independence

The approach students take to their own learning has been described in various studies, mostly from the perspective of active/independent learning or passive/dependent learning.

Dependence has been described by Baird and Mitchell (1991) as a poor learning tendency where the learner, despite being keen to succeed, adopts a passive, dependent approach to learning. The learner assumes the role of “a passive recipient of information and instructions”, expecting the teacher (or text) to tell them what to do and how to do it (Baird & Mitchell 1991; Carbone 2007).

In a study of programming students, dependence was highlighted as a deficit in generic skills by Carbone et al. (2009). When students got stuck, they sought assistance from experienced friends and tended to act on their suggestions, regardless of whether they had understood or not. One study that investigated student programming success from the instructor’s perspective, reported that teachers believe that success hinges on students taking responsibility for their learning and putting in sufficient effort (Kinnunen et al. 2007).

A model of “ownership of learning” is presented by Conley and French (2014) in their study of college readiness of students (see Figure 2-2). “Students who own their own learning can go beyond simply following teacher directions. They are more likely to complete complex assignments, solve problems that require persistence, and create original or novel work of high quality” (Conley 2014; Conley & French 2014).
The Conley and French (2014) model of ownership of learning consists of:

- **Motivation and Engagement**: Self-guided actions where the students see value in the coursework, are motivated to excel, are interested in the content and enjoy a challenge.

- **Goal orientation and self-direction**: Strategies to set academic goals and identify the resources and steps to attain them. This relies on the ability to exercise control over one’s behaviour.

- **Self-Efficacy and Self-Confidence**: Students’ confidence in their ability to complete increasingly challenging and complex academic and career tasks.

- **Metacognition and Self-Monitoring**: Awareness of the learning process, active participation in the learning and reflecting on that participation.

- **Persistence**: Related to resilience but also encompasses the notions of grit and academic tenacity.

In a study of undergraduate student approaches in peer-led workshops in science, engineering, technology and mathematics, three approaches to learning were identified (Micari & Light 2009):

- **Reliance** – Thinking about getting through the course:
  - Intentions: reducing anxiety and confusion and gaining confidence and clarity.
  - Constraints: Lack of confidence, self-discipline and time management skills.

- **Engagement** – Thinking about engaging with the material:
  - Intentions: reducing passivity and increasing involvement, applying rather than absorbing.
  - Constraints: Watching rather than doing, knowing principles but not know how to apply them, only attempting basic problems, more help is required.

- **Independence** – Thinking about how to learn:
  - Intentions: Becoming a better learner, developing an integrated system for understanding.
  - Constraints: Limited set of problem solving tools, concepts stand alone.
Self-Regulated Learning (SRL) has been an important topic in the fields of education and psychology. SRL is defined as the degree to which learners are metacognitively, motivationally and behaviourally active participants in their own learning (Zimmerman 2002). Studies have found SRL to have a significant positive correlation with academic achievement in elementary, high school, and college students (Bergin & Reilly 2006). One model of Self-Regulated Learning includes the following components (Pintrich 1999):

- **Cognitive Learning Strategies**: includes rehearsal, elaboration and organizational strategies
- **Metacognitive and Self-Regulatory Strategies**:
  - Monitoring strategies – setting goals for studying and doing task analysis of the problem.
  - Regulation Strategies – re-align back to the goal when the learner realises a breakdown in understanding. E.g. going back and re-reading the text that is not understood.
- **Resource Management Strategies**: controlling their time, effort, study environment, and other people such as peers and teachers through help seeking strategies.
- **Motivational Beliefs**: include self-efficacy beliefs, task value beliefs and goal orientations.

The cognitive and metacognitive strategies represent the “skill” component and the motivational beliefs represent the “will” component (Bergin & Reilly 2006).

Students who perform well in programming were found to use more meta-cognitive and resource management strategies than the lower performing students (Bergin, Reilly & Traynor 2005). Ott et al. (2015) attempted to improve student’s self-regulation during their learning by supplying them performance indicators at different stages of an introductory programming course. They found, however, that the students’ study behaviour and learning outcome remained unaffected, despite them valuing the information.

In a study by Sheard et al. (2013), students described a range of independence in their learning behaviours and willingness to take responsibility for their own learning. Most students described some degree of independence by expressing a desire to work alone and understand better by themselves. In contrast, other students described a desire to be taught while one student expressed preference for the type of teaching experienced in high-school where students rely on their teachers (Sheard et al. 2013).
2.2.4 Self-Efficacy

Self-efficacy is a person’s perception of their own ability to succeed in a performance situation. Self-efficacy theory has been used as a tool to understand and promote intrinsic motivation (Bandura 1997, 2002). “A person who perceives themselves to have a low-level of self-efficacy will, despite having the necessary potential, ability and skill, not perform well at carrying out a task” (Bomia et al. 1997). Self-efficacy beliefs influence the choices a person makes, such as the degree of effort spent on a task, the level of persistence and the response to obstacles (Bomia et al. 1997; Wiedenbeck, LaBelle & Kain 2004).

There are four principal influences on self-efficacy (Bandura 1997, 2006; Lin 2016):

- **Mastery Experiences**: repeated successes increase self-efficacy while repeated failures reduce it.
- **Vicarious Experiences**: Observing similar peers succeed can increase expectations of success.
- **Social Persuasion**: If an individual is persuaded they can complete a task, they try harder.
- **Physiological State**: Levels of stress, tension, fatigue, mood, pain etc. can impact self-efficacy.

Within the programming education context, a considerable number of studies have found a positive correlation between a student’s self-efficacy of programming and their performance in programming courses (Adair & Jaeger 2011; Altun & Mazman 2015; Bergin & Reilly 2005a, 2005b, 2006; Gomes, Santos & Mendes 2012; Kanaparan 2016; Özmen & Altun 2014). In general, students who feel capable of writing programs tend to perform well, while students who feel they are not capable of writing programs perform poorly.

Similarly, studies have found a correlation between “comfort level” and performance (Cantwell-Wilson & Shrock 2001; Ventura 2005). Students that feel comfortable with their studies will perform better, while students who feel anxiety about learning programming will fare worse.

A number of factors have been found to influence self-efficacy, which in turn influences performance. Several studies have found that previous programming experience is linked to programming self-efficacy (Özmen & Altun 2014; Wiedenbeck 2005; Wiedenbeck, LaBelle & Kain 2004). Another found that persistence levels had a significant effect on a student’s self-efficacy beliefs (Lin 2016). Students may also use comparisons with their classmates as a base for their self-efficacy perceptions (Kinnunen & Simon 2011).
Several studies have looked at gender influences on self-efficacy, with mixed results. Lin (2016) found no significant differences in self-efficacy between genders, while Beckwith et al. (2005) found females exhibited lower self-efficacy. Female students have also been found to revise their self-efficacy beliefs earlier than males, which suggests early failures could cause female students to disengage from their studies (Lishinski et al. 2016).

Variations in self-efficacy have also been studied. Wiedenbeck, LaBelle and Kain (2004) found that self-efficacy does increase substantially during an introductory programming course. Changes, however, are also not always straightforward. Students may reflect negatively on their self-efficacy after successfully completing an assignment, or positively after struggling with an assignment (Kinnunen & Simon 2011, 2012).

### 2.2.5 Mindset

Mindset Theory has been developed by Dwek (2008), and others, over the past two decades and describes how a person’s belief about their ability influences their response towards a goal (Cutts et al. 2010; Diener & Dweck 1980; Dweck 2000; Elliott & Dweck 1988; Robins, RW & Pals 2002). Dwek (2008) describes two mindset frameworks:

- **Fixed Mindset (learnt helplessness)**
  Failures are overestimated and seen as unavoidable, recurring and unconquerable. Intelligence is seen as static so effort is placed in appearing smart rather developing skills. Failure is viewed as resulting from lack of intelligence with decreasing self-esteem and the response to setbacks is to give up. The judgement of others and self is seen as important (Diener & Dweck 1980; Dwek 2008).

- **Growth Mindset (mastery orientation)**
  Failures are seen as conquerable, avoidable and successes replicable. Intelligence can be developed, so effort and time is spent on developing skills. Failure is attributed to lack of effort thus maintaining self-esteem. Learning and helping others learn is important (Diener & Dweck 1980; Dwek 2008).

Fixed mindset learners adopt performance oriented goals while growth mindset learners adopt learning oriented goals (Elliott & Dweck 1988). The learning tendencies of each type of mindset are summarised in Table 2-1 (Dwek 2008):
A number of studies have recognised the importance of self-theories (mindset) research and how this relates to and impacts the learning of programming. Programming students are confronted with an excessive amount of challenges and negative feedback when learning to program for the first time. Students with fixed mindsets are likely to adopt a helpless response to these innate challenges and view difficulties as a lack of ability or intelligence and disengage by practicing “risk avoidance and strategy abandonment” (Murphy & Thomas 2008).

A study by Cutts et al. (2010) implemented several mindset based interventions: “tutors taught mindset to students; growth mindset feedback messages were given to students on their work; and, when stuck, students were encouraged to use a crib sheet with pathways to solve problems”. They found that these changes resulted in a significant change in mindset and improvement in test scores. They did recommend that since programming students come up against failure more often than in other subjects, the mindset message may need to be reinforced regularly (Cutts et al. 2010).
Hanks et al. (2009) asked students what advice they would give to future students to develop their growth mindset towards programming. Some of the advice noted by students included:

- **Make the most of the class:** Clear doubts immediately and get help when you need it. It is very hard to catch up in programming if you fall behind. Programming is cumulative.
- **Learn from mistakes:** The key to understanding is using each error as a stepping stone to learn and understand why the error occurred. “Everybody makes mistakes – learn from them”.
- **Learn from others:** “If you cannot work out your mistake, find someone to help you, or discuss with your friends”.
- **You have to program to learn:** “After solving a problem, change the code and play around with it. Experimenting is the best way to learn”.
- **Have a plan:** “1. Read through the descriptions of the program and make sure you know what you are supposed to do. 2. Make a flow chart. 3. Write the program step by step.”
- **Don’t be intimidated:** “Don’t be afraid of asking questions in class” – “Do not worry about looking stupid for not knowing” – “other students may be wondering the same thing”
- **Programming can be fun:** Learning programming is a challenge but also an opportunity. Programming is fun when you start to know it (Hanks et al. 2009).

Mindsets may also be domain specific. Scott and Ghinea (2014) found that students’ mindset for programming aptitude was different to their mindset about intelligence. Educators, therefore, should focus specifically on promoting growth mindsets for programming.

Studies have also looked at the relationship between self-theories (mindsets) with resilience and grit (passion and perseverance to pursue a goal over a period of years). Mindsets have been found to influence resilience (Scott & Ghinea 2014; Yeager & Dweck 2012). A study of high school students by Kench, Hazelhurst and Otulaja (2016) found a weak correlation between mindset and grit.
2.2.6 Emotions

Some recent studies have tried to understand student emotional experiences when learning to program. A study by Kinnunen and Simon (2010) investigated students' emotional experience with and reactions to programming assignments. They identified emotions at six stages of the programming experience:

- **Getting Started**: “It’s Greek to me”, “OK, what now?”
- **Encountering Difficulties**: Hit by lightning experience, Rapid change experience
- **Dealing with Difficulties**: Feedback guided experience, Hamster wheel experience, Course-specific experience, other computing factors experience
- **Overarching**: Life outside this assignment experience, Helping others experience
- **Auxiliary emotional load**: Impact of outside factors
- **Self-Efficacy Experience**: At all stages, there were emotional consequences from students assessing their ability compared to others or to academic success.

One category highlighted in their findings was the “hit by lightning” experience, which they labelled as the mark of the utter novice. This is where a student encounters a problem that comes out of blue, impacts their confidence and leaves them with little idea of what is wrong or how to fix it (Kinnunen & Simon 2010).

Bosch, D'Mello and Mills (2013) got students to self-report their emotions during programming learning sessions. They found that the most common emotions students experienced were: flow/engaged (23%), confusion (22%), frustration (14%) and boredom (12%). On the other hand, curiosity, happiness, anxiety, surprise, anger, disgust and fear were rare. Chetty and van der Westhuizen (2013) found that novice students experience strong oscillating emotions while learning to program, especially when learning threshold concepts.

2.2.7 Summary

Recent research into programming student motivations has given important insights, but more exploration is necessary. With programming now being taught to a much wider range of students, a deeper understanding is needed of how these students perceive programming as relevant to their careers and how that influences their motivations. With much of the learning now happening beyond the classroom, a deeper understanding is also needed of how independent and dependant learning impacts student learning of programming.
2.3 Learning Behaviours

How students go about their learning has been widely researched from various perspectives. Deep and surface learning is a key perspective taken by many researchers. Another view has been to look at good and poor learning tendencies. Several models of how students experience the learning of IT and programming have also been developed. Yet another perspective is that of student engagement: behavioural, emotional and cognitive. This section presents these various perspectives of learning behaviours.

2.3.1 Deep and Surface Learning

Various authors have sought to explore how students learn from the perspective of surface learning and deep learning. While the majority of these studies (and models developed) are from a general education perspective, a few studies have focussed specifically on the Information Systems, Computer Science and Accounting contexts.

A widely used model in the field of education is the Biggs 3-P Model of Learning (see Figure 2-3), that depicts that the interactions between student factors and the teaching context produce an approach to learning by the student which yields a particular learning outcome (Biggs & Tang 2007).

![Figure 2-3 - Biggs 3P Model of Learning (Biggs & Tang 2007)](image-url)
Biggs contends that the approaches to learning are formed by motives (intention) and strategies (actions). It is the learning approach the student chooses (surface, deep or achieving) that affects the quality of the learning outcome (Biggs 1989; Biggs & Tang 2007, 2011; Carbone 2007):

- **Surface Approach**

  The surface approach is based on an extrinsic motivation such as choosing to do a degree at university as a means to some other end such as obtaining a better job or to achieve a pass as a result of having to take a subject that is irrelevant to the student’s programme. This approach arises from the intention to get the task out of the way with minimum effort while appearing to meet the task requirements. It involves ‘cutting corners’ to give the impression that the task was properly done, this is to avoid failure. Students focus on the isolated items and treat these independently of each other or to other tasks. They focus on the literal facts missing their meaning (rote learning). There is also a presence of negative feelings such as anxiety, cynicism and boredom. There may be other factors that encourage a surface approach such as: Insufficient time, misunderstanding requirements, high anxiety, or a cynical view of the subject or of the teaching context itself.

- **Achievement/Strategic Approach**

  The achieving approach is based on a particular type of extrinsic motivation: the need to achieve through high grades using a surface approach. The strategies used consist of organizing time, working space and syllabus coverage in the most cost effective way. Their approach is systematic, planning ahead and allocating time to tasks in proportion to their earning potential.

- **Deep Approach**

  The deep approach is based on an intrinsic interest in the subject matter that is being learnt. The strategy used aims to maximise understanding so that curiosity is satisfied. Students feel the need to engage in the task appropriately and meaningfully. They feel a need-to-know so they focus on understanding the underlying meanings rather than the literal aspects. They seek to integrate the parts with other tasks and also with the big ideas in the domain.

  The student reads widely, discusses with others, and may play with the task in order to hypothesise how it relates to other known or interesting items. Students generally have positive feelings when using a deep approach such as feelings of interest, challenge, exhilaration. Factors that promote a deep approach include an intrinsic curiosity or determination to do well, appropriate background knowledge and a well-structured knowledge-base.
A model more specific to Information Systems was developed by Cope (2000), who sought to investigate the relationship between approaches to learning and levels of understanding of the concept of an Information System (IS) in undergraduate students (Cope 2000, 2002, 2003).

- **Surface Approach**

A surface approach to learning about IS involved a series of isolated and generally unrelated learning and assessment tasks that had to be completed to pass the course. Knowledge was experienced as facts contained in lecture notes and summaries, formulae and skills on how to use software. The intentions of acquiring the knowledge were to be able to recall the knowledge or to apply the memorized procedures in assessment situations.

- **Deep Approach**

A deep approach to learning about IS involved seeking meaning of the content from the learning tasks and assignments with the intent of developing a deep personal understanding that went beyond the immediate task. Meaning was sought through relating the content in the learning tasks to other content from the same course or a different course. Understanding was experienced as having an internalised picture of the content that could be used to link with new material or explain to others.

Cope (2003) described that the educationally critical characteristics that lead to the development of a deep understanding are:

- An intention to seek a deep understanding of the concept of an IS.
- The process of seeking and relating the meanings associated with a broad range of different perspectives on the concept of an IS, including perspectives in personal experiences beyond the academic setting and in studies outside IS and computing courses.
- An awareness of one’s own understanding of the concept of an IS and the approaches being used to learn about IS.

Learning activities should centre on students being aware of and taking charge of their own approaches to developing a deep understanding of the concept of an IS (Cope & Prosser 2005).
While the Biggs model has been widely cited in computing education literature, few studies were found that specifically investigated deep and surface learning approaches in the programming context. One major study conducted over multiple universities, in multiple countries, of introductory programming students, found that a deep approach to learning was positively correlated with marks in programming while a surface approach had a negative correlation to marks (de Raadt et al. 2005; Simon et al. 2006).

Another study investigated deep, surface and strategic learning approaches in programming students. They found that a strategic approach had a strong positive relationship with performance, a surface approach had a strong negative relationship, but a deep approach only had a weak positive influence. They conclude that the students who take a strategic approach are most likely to be successful (Hughes & Peiris 2006).

### 2.3.2 Learning Tendencies

An alternative view to learning approaches is the concept of learning tendencies (Baird & Mitchell 1991; Baird & White 1982). Baird and Mitchell (1991) describe poor learning tendencies as “descriptions of poor thinking habits – of deficiencies in the way learners tackle tasks or process information”. Baird and Mitchell found that poor learning tendencies were frequently responsible for poor learning performance, but often unintended and that students were unaware of them, even by those with a good level of commitment.

Carbone (2007) used these poor learning tendencies as a “base for determining the type of thinking undergraduate IT students did not do as they engaged in their programming tasks”. She developed the poor and good learning tendencies depicted in Table 2-2 and Table 2-3.

Carbone (2007) also found that programming students can suffer from poor time management. Failed attempts to complete a working system within the timeframe allowed led students to prioritise the activities to complete and choose those that would achieve the best grades possible. They also abandoned their attempts to understand the problem and opted to copy slabs of code from the textbook without fully understanding it (Carbone 2007; Carbone et al. 2009).
<table>
<thead>
<tr>
<th>Learning Tendencies</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Superficial Attention</td>
<td>This involves skimming over a communication, with no attempt to actively process the information in order to generate personal meaning.</td>
</tr>
<tr>
<td>Impulsive Attention</td>
<td>Some parts of a communication that appear interesting are attended to but others that may be major points are overlooked.</td>
</tr>
<tr>
<td>Staying Stuck</td>
<td>Lack of any strategy to cope with getting stuck except to call for help. No attempt to return to the instructions, reflect on the strategy selected, analyse what has been done so far or consider alternative approaches.</td>
</tr>
<tr>
<td>Premature Closure</td>
<td>Ceasing work on a task in the genuine belief that it is finished when in fact some things may not have been done.</td>
</tr>
<tr>
<td>Inappropriate Application</td>
<td>Blind application of a memorised procedure in a situation where it is not applicable.</td>
</tr>
<tr>
<td>Non-retrieval</td>
<td>This occurs when no attempt is made to retrieve one’s own existing views and understandings which are relevant to the knowledge being presented. The learner is unsure of conflicts between the school knowledge and their personal views.</td>
</tr>
<tr>
<td>Lack of Internal Reflective Thinking</td>
<td>The learner is not thinking reflectively about the content as presented “internally” (i.e. within the boundaries of the subject). Each lesson, activity or even instruction is seen as isolated from the others.</td>
</tr>
<tr>
<td>Lack of External Reflective Thinking</td>
<td>The learner makes no attempt to link the content of one subject with the content outside the boundaries of the subject (i.e. world or other subjects).</td>
</tr>
<tr>
<td>Ineffective Eradication</td>
<td>Persistent reappearance of apparently changed misconceptions or alternative explanations.</td>
</tr>
</tbody>
</table>

Table 2-2 - Poor Learning Tendencies (Baird & Mitchell 1991; Carbone 2007)
<table>
<thead>
<tr>
<th>Summary</th>
<th>Possible Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seeks Assistance</td>
<td>Informs teacher when they don’t understand.</td>
</tr>
<tr>
<td></td>
<td>Asks teacher why they went wrong.</td>
</tr>
<tr>
<td></td>
<td>Informs teacher what they don’t understand.</td>
</tr>
<tr>
<td>Checks Personal Progress</td>
<td>Checks work against instructions, correcting errors and omissions.</td>
</tr>
<tr>
<td></td>
<td>Refers to earlier work when stuck before asking the teacher.</td>
</tr>
<tr>
<td></td>
<td>Checks personal comprehension of instruction and material.</td>
</tr>
<tr>
<td>Plans and Anticipates</td>
<td>Anticipates and predicts possible outcomes.</td>
</tr>
<tr>
<td></td>
<td>Plans a strategy before starting.</td>
</tr>
<tr>
<td></td>
<td>Explains purpose and results.</td>
</tr>
<tr>
<td>Reflects on work</td>
<td>Checks teacher’s work for errors and offers corrections.</td>
</tr>
<tr>
<td></td>
<td>Seeks links between adjacent activities and ideas.</td>
</tr>
<tr>
<td></td>
<td>Seeks links between different topics.</td>
</tr>
<tr>
<td></td>
<td>Independently seeks further information.</td>
</tr>
<tr>
<td>Links to Beliefs and Experiences</td>
<td>Seeks links between different subjects.</td>
</tr>
<tr>
<td></td>
<td>Asks inquisitive but general questions.</td>
</tr>
<tr>
<td></td>
<td>Offers personal examples which are generally relevant.</td>
</tr>
<tr>
<td></td>
<td>Seeks specific links between school work and personal life.</td>
</tr>
<tr>
<td></td>
<td>Searches for weakness in their own understandings.</td>
</tr>
<tr>
<td></td>
<td>Checks the consistency of their explanations across different situations.</td>
</tr>
<tr>
<td>Assumes a Position</td>
<td>Suggests new activities and alternative procedures.</td>
</tr>
<tr>
<td></td>
<td>Expresses disagreement.</td>
</tr>
<tr>
<td></td>
<td>Offers ideas, new insights and alternative explanations.</td>
</tr>
<tr>
<td></td>
<td>Justifies opinion.</td>
</tr>
<tr>
<td></td>
<td>Reacts and refers to comments by other students.</td>
</tr>
<tr>
<td></td>
<td>Challenges the text or an answer the teacher sanctions as correct.</td>
</tr>
</tbody>
</table>

Table 2-3 - Good Learning Tendencies (Baird & Mitchell 1991; Carbone 2007)

### 2.3.3 Ways students experience learning IT and Programming

Other studies in the way students approach their learning in the fields of Computer Science (CS) and IT have demonstrated that students experience their learning in different ways.
Berglund’s et al. study presents seven categories which describe the way students act to learn Computer Science, where the more advanced categories with a higher number describe a more sophisticated way of acting to learn. Categories 1 to 6 describe learning CS through (Berglund & Wiggberg 2008):

- **1- Learning to use application programs**, focussing on learning the tools that are used and neglecting the other aspects in the field taught by the computer science community.
- **2- Learning CS through learning about isolated concepts**, focussing on learning concepts/topics of CS rather than what can be done with what is learnt.
- **3- Learning CS through consolidating what is already known**, where deeper insights about CS concepts are sought but concepts are still seen in isolation.
- **4- Learning CS through analysing systems**, focussing on learning the whole in order to explore the different concepts which make up the parts.
- **5- Learning CS through integrating systems**, focussing on learning by splitting the units in the software system into pieces and analysing the parts and then putting the components together, as a way to learn about and to create a system.
- **6- Learning CS through giving meaning to concepts**, focussing on seeking personal insights and personal meanings to the CS concepts.
- **7- Learning CS through developing as a professional**.

Peters et al. (2014) phenomenographic study focused on understanding the distinct ways students participate in CS and IT using the idea of participation from Wenger’s social theory of learning. Participation in CS/IT was experienced as:

- **Using**: to make use of what exists for various purposes
- **Inquiry**: activities that aim at understanding, learning, informing
- **Creating things**: to produce things that were not there before. The aspects that are related with this category are: the outcome, the process of doing and doing with others
- **Systematic problem solving**: This includes using methods, ways of thinking and systematically working with others to create things
- **Creating for others**: This includes taking into account the user’s perspective in the process of creating and problem solving
- **Continuous development**: as a continuous process of improvement
- **Creating Knowledge**: to develop new solutions, (to do research)
Introductory programming courses often have highly bimodal grades – many students failing or obtaining high grades. Robins (2010) proposes this may be due to what he calls “Learning Edge Momentum” (LEM). Concepts in the programming domain are tightly integrated, where almost every concept depends on many others. Successful learning of a concept makes it easier to learn further related concepts. In contrast, unsuccessful learning of a concept makes it harder to learn other closely linked concepts. This then creates a momentum towards a successful or unsuccessful outcome (Robins, A 2010).

Another factor that explains the LEM effect is that “we learn at the edges of what we already know”. “Understanding (on a short time scale) and learning (over a longer period) depends on fitting new material into the context of existing knowledge” (Robins, A 2010).

Bruce et al. (2004) investigated how first year university students experience the act of learning programming. The experiences are described as five different categories: Following, Coding, Understanding and Integrating, Problem Solving, Participating (see Figure 2-4). These categories are further elaborated in section 2.6.

![Figure 2-4 - Ways students experience programming model (Bruce et al. 2004)](image)
2.3.4 Engagement

Engagement has been described as a multidimensional construct that integrates and blends the concepts of behaviour, emotion and cognition (Fredricks, Blumenfeld & Paris 2004). Engagement has been associated with positive academic outcomes, such as achievement and persistence in school. It has also been found to be higher in classrooms with “supportive teachers and peers, challenging and authentic tasks, opportunities for choice, and sufficient structure” (Boekaerts 2016; Fredricks, Blumenfeld & Paris 2004).

Three different types of student engagement have been described (Carbone 2007; Fredricks, Blumenfeld & Paris 2004):

- **Behavioural Engagement**
  - Positive conduct, such as following rules and adhering to classroom norms, as well as absence of disruptive behaviours such as skipping school and getting into trouble.
  - Involvement in academic tasks which includes behaviour such as effort, persistence, concentration, attention, asking questions and contributing to class discussion.
  - Participation in school related activities such as athletics or school governance.

  Behavioural engagement contributes towards positive academic outcomes.

- **Emotional Engagement**

  Students’ affective reactions in the classroom towards teachers, classmates and school – both positive and negative. These include interest, boredom, happiness, sadness and anxiety and are considered to influence the willingness to do work. These emotions overlap with many of those reported in the motivational literature.

- **Cognitive Engagement**

  “Effort directed towards learning, understanding, mastering the knowledge, skills or crafts that the academic work is intended to promote” (Fredricks, Blumenfeld & Paris 2004). Cognitively engaged students show flexibility in problem solving and the ability to cope with failures. They use deep level strategies, exerting greater efforts to achieve a more thorough understanding. They want to go beyond the requirements and prefer a challenge.
The concept of engagement has been widely studied in recent years (Boekaerts 2016; Eccles 2016). Azevedo (2015) found more than 32,000 articles in PsychInfo about engagement published in the last 14 years. Researchers have broadened the original three dimensional framework of engagement to include agentic, self-regulatory and social-behavioural components (Boekaerts 2016). Influences on engagement have also been explored, such as teaching staff, peers, structure and task characteristics (Fredricks 2011).

In a recent study of engagement in first year ICT students, Sheard, Carbone and Hurst (2010) used the three types of engagement as a framework to analyse the data. Low behavioural engagement was identified as poor class attendance and lack of weekly study time, with students spending half of the recommended time. They also displayed a lack of organisational ability where they put low effort at the start, which must be increased when assignments are due. Irregular study patterns may be also influenced by outside work commitments (Sheard, Carbone & Hurst 2010).

Low cognitive engagement manifested itself as a low involvement and investment in study time. While teachers expected students to work more independently and to read the lecture notes prior the lecture, few did. Students, on the other hand, expected more guidance, concrete examples to learn from and specific topics to learn for the exam (Sheard, Carbone & Hurst 2010).

Affective engagement was related to the level of interest students had in the course content. Students were keen to learn the applications but not that interested in fundamental concepts. Differing levels of ability were found to influence engagement. Highly engaged students have an interest in the area and have already learnt some ICT on their own or from high school. In contrast, lower engaged students failed to attend classes and avoided consulting teaching staff (Sheard, Carbone & Hurst 2010).

2.3.5 Learning Outside the Classroom

With the wide adoption of learning management systems and course materials being accessible online, students now have much greater flexibility in choosing the mode, place and time of their learning (Sheard et al. 2013). Relatively few studies, however, have investigated programming student study habits outside the classroom.
In a survey of programming student study habits, Chinn et al. (2010) found that “students engaged in a wide range of study behaviours in terms of time spent and use of resources”. Students spent as much time accessing the Internet as reading the textbook. While the majority of their students said they preferred to study alone, they also reported that 95% of students spent at least some time each week talking to friends and classmates about the unit. Their female students tended to work more with others than the male students and they reported this as being a negative factor on their results (Chinn et al. 2010; Sheard et al. 2013).

A follow up study by Sheard et al. (2013) sought a more holistic understanding of the study habits of programming students. Students reported studying in various locations, such as at home, on campus, or on the train. Many students reported spending minimal study time after tutorials and then spending intense periods of study time during assignment /exam time.

Students reported using various study techniques outside the classroom. Most reported reading lecture notes, summary notes, and the textbook (as a reference source or for code examples). Many also heavily relied on online materials. Most students also reported attempting lab and tutorial exercises outside of class time but only some tried running code examples. Students found that working with others was helpful, however, most preferred to work on their own (Sheard et al. 2013). In a recent study of programming study habits, Willman et al. (2015) found that “students who receive the highest grade start and finish their work early, do not work on weekends, and do not work at night”.

More investigation of how students study outside of the classroom is required to better understand and help students in their efforts to learn programming.

2.3.6 Summary

This section presented various perspectives of student learning behaviours: deep and surface learning, good and poor learning tendencies, how students experience the learning of IT and programming and student engagement. Much of this research, however, has been focussed on how students learn inside the classroom. Students today have access to their learning materials at any time and place they choose. More understanding is needed on how students are learning programming outside of the classroom.
2.4 Problem Solving

Many studies have researched the difficulties and obstacles programming students are challenged by in problem solving. The majority of these studies have focussed on novice, first year, programming students (Adair & Jaeger 2011; de Raadt 2007; Malik & Coldwell-Neilson 2016; Pillay & Jugoo 2005; Robins, A, Rountree & Rountree 2003; Sarkar et al. 2013).

Programming has been described as being problem-solving intensive, “requiring a significant amount of effort in several skill areas” (Jenkins 2002). Students have difficulty learning, in a short period of time, the multiple sets of skills and processes required to translate a specification into an algorithm and then into program code (Adair & Jaeger 2011). They are often faced with needing to learn new problem solving approaches that are different to the study and problem solving approaches they are already familiar with (Jenkins 2002; Perkins, Schwartz & Simmons 1988). Students themselves also report that understanding problem solving strategies is one of their major learning difficulties (Malik & Coldwell-Neilson 2016; Piteira & Costa 2013).

A study by McCraken et al. (2001) of 200 novice programming students from four institutions in three countries found that after one or two semesters of programming studies, students still had difficulties with problem solving. They described five steps of problem solving: abstracting the problem from its description, generating sub-problems, transforming sub-problems into sub-solutions, re-composing the sub-solutions into a working program, evaluating and iterating.

There have been a wide range of problem solving behaviours reported in the literature, and these are summarised in this section.

2.4.1 Use of Strategies

Novices have been found to have difficulties determining strategies to use for problem solving, being able to apply strategies correctly, translating an algorithm into code and decomposing a problem into sub-problems (Lahtinen et al. 2005; Özmen & Altun 2014; Robins, A, Rountree & Rountree 2003; Soloway 1986; Winslow 1996). Also, they dive straight into the coding part of programming without a clear understanding on the problem or a systematic strategy to solve it (Carbone 2007; Carbone et al. 2009). They tend to use general problem solving strategies (such as copying a similar solution or working backwards from the goal) instead of programming specific strategies (Winslow 1996).
Perkins et. al. (1989) identified three types of students and the strategies they use when attempting to correct semantic problems in their code. Stoppers will give up solving a difficult problem or ask for help. Movers will tend to systematically go through the problem using sufficient code tracking to solve it. Tinkerers, however, will tinker with their code without sufficient tracking and with little grasp of why the program behaves as it does. They assume minor changes will help when in fact they need to take a different approach (Perkins et al. 1989).

Students who practice “ineffective tinkering” use trial and error without a plan and proceed to iteratively modify and recompile problem code compounding the problem rather than fixing it (Carbone et al. 2009). Murphy et al. (2008) described different types of tinkering that students practiced in order to fix code problems, including the “just in case” strategy where unnecessary changes are done, such as deleting unused variables. “Work around the problem” is another form of tinkering where code that is not understood is replaced with new code that is less elegant (Murphy et al. 2008). Students also fail to recognise the need or are unable to break a programming problem into parts to test each part separately (Carbone et al. 2009; Perkins et al. 1989).

Successful students have discussed the strategies they use to get unstuck (McCartney et al. 2007).

- The “inputs/interaction” strategies consist of students learning from other people, tools or written materials and by following step-by-step instructions. Students interact by discussing, learning from peers, listening to teaching staff and by getting help.
- The “concrete/do stuff” strategies consist of gaining experience by practicing and learning from examples, combined with getting help from others. Practicing includes writing programs, learning from trial and error, practicing additional drills on their own and tracing.
- “Abstract/understand stuff” strategies consist of relating concepts to real world examples which includes breaking a problem into parts and using incremental development; seeing the context or reason or use for something, seeing the large picture and identifying patterns.
- “Use the force” strategies consist of students using their will power or character such as students telling themselves to persevere or taking a break and coming back to the problem.

Sheard et al. (2013) found that some students rely on a model solution to work from, learning by rote or rely on teacher directed learning. The more successful students choose their preferred level of granularity to look at the code, as a way of working through a problem.
2.4.2 Planning

Strategies describe how the programming knowledge is applied, and are made up of plans, schemas or patterns which are associated and put together into a single solution (Robins, A, Rountree & Rountree 2003). However, it is in the “putting the pieces together”, “composing and coordinating components of a program” where novices have problems (Soloway 1986).

de Raadt, Toleman and Watson (2004) tested a group of novice programming students on their ability to make simple programming plans at the end of their semester. They found mixed results and concluded that students cannot be expected to implicitly learn planning strategies and that they should be taught explicitly (de Raadt, Toleman & Watson 2004).

A key difference between novices and experts lies in the experts’ ability to plan (de Raadt 2007). Experts refer to their “gut feeling and intuition” when using their knowledge and strategies to solve a problem. Experts are not necessarily conscious that they are relying on tacit knowledge built up from solving past problems (Soloway 1986). A study by de Raadt, Watson & Toleman (2006) looked at how experts solved novice level type of programming problems. The experts’ solutions showed consistent use of anticipated plans.

The seminal work by Spohrer and Soloway (1986) described nine kinds of plan composition problems: (Summarization, Optimization, Previous Experience, Specialization, Natural Language, Interpretation, Boundary, Unexpected cases and Cognitive Load problems). They also found that students have difficulties in using multiple plans together (abutting, nesting, tailoring and especially merging).

2.4.3 Code Comprehension

There have been various studies that have looked at the novice student’s ability to trace through code and understand it. It has been widely reported that novices tend to have difficulties reading code and comprehending what the program does (de Raadt 2007; Lister et al. 2004; Robins, A, Rountree & Rountree 2003; Rountree, J et al. 2005).
“Many students lack knowledge and skills that are a precursor to problem-solving” and these missing skills often relate to the ability to read code as well as writing it (Lister et al. 2004). Novices have been found to perform very poorly on standardized program generation tests, but, comparatively, they performed better in program comprehension tests but not as well as expected by instructors (de Raadt 2007). This may indicate that novice programming knowledge is fragile and may be partly responsible for poor generation tests (de Raadt 2007).

Ahmadzadeh, Elliman & Higgins (2005) found that even good students lacked the ability to read and understand the actual program implementation (i.e. especially other people’s code) as well as having a lack of knowledge of error and of debugging strategies. Özmen and Altun (2014) found that students have difficulty understanding and reading code, including changing their own previously written code. They also had difficulties understanding the semantics of the program regardless of their programming knowledge (Özmen & Altun 2014). In contrast, Sheard et al. (2013) described students looking at a solution to a program before writing the code.

2.4.4 Fragile Knowledge

Perkins and Martin (1986) described the concept of fragile knowledge as students having knowledge and perhaps being able to articulate it, but being unable to apply that knowledge when required to write a program. The student “sort of knows, has some fragments, can make some moves, has a notion, without being able to marshal enough knowledge with sufficient precision to carry a problem through to a clean solution” (Perkins & Martin 1986). They describe the different manifestations of fragile knowledge as:

- Partial Knowledge: Knowledge that is either missing, never learnt or forgotten.
- Inert Knowledge: The student fails to retrieve and use knowledge that has been learnt.
- Misplaced Knowledge: It has been learnt but it is used inappropriately.
- Conglomerated Knowledge: Separate elements are merged together in a syntactically or semantically unusual way.

Fragile knowledge of programming has also been put forward as a possible explanation as to the reason novices struggle to problem solve, by a number of other authors (de Raadt 2007; de Raadt, Watson & Toleman 2006; Lister et al. 2004; Robins, A, Rountree & Rountree 2003).
2.4.5 Tracking Code Difficulties

In general, the actual implemented program can turn out to be different from the model of the program that was intended. These discrepancies cause an executing program to have unpredicted outcomes, and there may also be bugs present. To identify these problems, programmers need to understand the program that is running (Robins, A, Rountree & Rountree 2003). Perkins et al. (1989) describe this ability as “close tracking”, which involves tracking/tracing the code in order to understand the executing program and predict its behaviour. Building an understanding or mental model of the executing program is essential for program comprehension, planning, testing and debugging phases in program generation (Robins, A, Rountree & Rountree 2003).

Novice programmers not only have difficulties translating the program model they intended into a computer program (Winslow 1996) but also they have difficulty tracing the execution logic of the code and identifying coding errors (Perkins et al. 1989). Even when they are aware of tracing and tracking strategies they neglect to use them (Perkins & Martin 1986).

Murphy et al. (2008) explored the different ways (and effectiveness) in which students trace/track their code. Their study found students trace mentally (by comparing the output with the code), on paper, using print statements or using a debugging tool to follow the flow of control of the program. The more effective methods involved printing the values of variables as execution progressed or using a debugger to step through the code and check the value of variables (Murphy et al. 2008).

Students require assistance to track and isolate code problems as the repeated “tinkering” attempts to correct the code tends to be incorrect (Carbone et al. 2009).

2.4.6 Code Writing Difficulties

It has been widely reported that novices tend to find writing code for programs difficult. “Novice programmers know the syntax and semantics of individual statements but they do not know how to combine these features into valid programs” (Winslow 1996). They tend to approach programming through control structures and use a line by line, bottom up approach to problem solution rather than using meaningful program “chunks” or algorithms (Winslow 1996).
According to Robins et.al. (2003) review on Rist’s (1995) work, “Experts can typically retrieve relevant plans from memory, and then generate code from the plan in linear order (from initialisation, to calculation, to output).” In contrast, novices must typically create plans and their code generation process begins with the central calculation first, then they build initialisations and other elements around it.

Students avoided going over, repeating or practicing programming which impacted on the programming knowledge knowing and remembering syntax while writing a program (Özmen & Altun 2014). The programming knowledge and code writing difficulties they identified were: remembering functions, parameters, concepts and principles related to the programming language, assigning a variable, choosing appropriate decision structures and loops.

2.4.7 Debugging Strategies and Difficulties

Debugging, the process of fixing coding problems (i.e. bugs) is one of the difficult programming phases novice students must contend with because it involves learning how to use multiple skills all at once (Murphy et al. 2008). Novice debuggers must apply many new skills simultaneously which include (Kessler & Anderson 1986):

- Understand the intended program and the execution of the actual buggy program
- Have general programming expertise
- Have an understanding of the programming language
- Comprehend the application domain
- Have knowledge of bugs and debugging methods

Coding errors can be syntactic (i.e. errors related to the use of the specific programming language or typos) or semantic (i.e. coding errors related to the way the logic unfolds making the program behave in an unintended way).

Some studies have reported that even good students, who have a good understanding of programming, have problems with debugging programs effectively (Ahmadzadeh, Elliman & Higgins 2005) (Rodrigo et al. 2013). In Rodrigo et al.’s study (2013) an analysis of syntactical errors revealed that all groups of students (low achieving, average and high achieving) struggled to a similar degree. However, the high achieving and average students resolved syntax errors more effectively, usually on the first try.
Repeatedly ineffective tinkering without sufficient understanding of the program’s actual misbehaviour generates multiple compilation errors where students fail to see error messages as vehicles for insight or learning (Carbone et al. 2009). Rodrigo et al. (2013) found the low achieving students spent their time battling through layers of additional syntactical errors added through ineffective tinkering attempts.

A lack of skills and strategies in using the debugger to fix compilation errors can hinder students’ ability to complete the task (Carbone et al. 2009). Problems in understanding the compiler messages on errors and also make debugging difficult (Özmen & Altun 2014).

Several studies have looked at identifying debugging strategies and how these are used by students. According to McCauley et al. McCauley et al. (2008), the study by Ducasse and Emde (1988) concentrated on the strategies used to locate bugs. They identified four global strategies that students use alone or in combination while debugging:

- Filtering (exploring program execution by tracing algorithms, scenarios and path rules)
- Checking computational equivalence of the intended program and the implemented program
- Checking the well-formedness of the actual program (language consistency checks)
- Recognizing stereotyped errors - those that can be easily recognised from experience

In Murphy’s et al. (2008) study some students were found to “use few strategies, applied them ineffectively, or engaged in other unproductive behaviours which led to frustration for some, and occasionally the introduction of new bugs”. They described various debugging strategies that students use, including tracing of code and tinkering, which are discussed earlier in this section. Additional debugging strategies described by Murphy et al. (2008) include:

- Gain Domain Knowledge - gaining an insight into the problem by re-reading the specification or re-examining the sample output.
- Understanding the Code - reading the code and understanding what happens to the variables.
- Testing - using sample data to test the program correctness. Very few students used calculators to compare to the program output, or tested with additional data such as to test boundaries.
- Using Resources – using online documentation, tutorials and/or old programs.
- Isolating the Problem - commenting out or the altering the code to isolate a problem. Sometimes correct code was commented out or changed incorrectly.
- Pattern Matching - recognising when things did not look right (syntax errors or missing braces).
2.4.8 Compilation Behaviours

A number of studies have attempted to look for patterns of student compilation and debugging behaviours by studying how students use the code compiler through analysing the compiler logs. One major limitation of these studies, noted by the authors themselves, is that data collection method does not help understand whether the student discovered and fixed the error themselves or whether they asked a friend or instructor for help (Jadud 2006).

Jadud’s (2006) study of compilation errors focused on the edit-compile cycle of novice programming students as they attempted to fix syntactical errors in their Java programs. They found that students often spend only a few seconds tweaking and then recompiling their programs. They also documented a common behaviour they called “remove the error”, whereby students removed problematic code lines in order to fix a syntactical problem, often unsuccessfully.

They also found that students in their study tended to write lots of code in one sitting, and then tried to correct all the syntax errors at once (Jadud 2006). This was also found to be a common practice in Carbone’s (2009) study where the whole solution would be typed up without any testing along the way, generating a series of compilation errors.

Later studies have focussed on the compilation logs captured by the Blackbox project, which provides a massive dataset of student usage of the BlueJ programming environment. A study by Altadmri and Brown (2015) looked at the frequency, spread of errors and the time it took to fix different Java errors. The analysis of errors was decoupled from compiling errors. Their findings suggest that semantic errors are more serious challenges than syntax errors. Jadud and Dorn (2015) attempted to develop a behavioural measure of how well novice programming students deal with syntactic errors. This study has not reported any findings as yet.

2.4.9 Summary

This section presented previous studies that have sought to understand how students go about problem solving when programming. These have included trying to understand how students use problem solving strategies, how they plan, how well they read and understand code, how well they track code execution, problems they have with writing code, problems they have with debugging and analysing compiler logs looking for patterns.
2.5 Social Networks and Information Behaviour

How programming students interact with others and seek information has a major impact on their learning. In the programming context, studies have looked at various forms of collaborative learning. Students form social networks and many of these are now via social networking platforms, but there has been limited research into how programming students use these platforms. There have also been limited studies into how programming students seek information and what information sources they make use of.

2.5.1 Collaborative Learning

Collaborative learning, or peer learning, involves students not only learning together but making students responsible for not only their own learning but of the learning of the group. Educators have applied a range of peer learning strategies to novice programmers, such as pair programming, peer testing, peer review of code and other forms of peer instruction. These have been shown to improve both individual performance and retention rates (Gaspar et al. 2013; Gaspar et al. 2016).

Collaborative learning is perceived by students to have a positive influence in the learning of programming, making it “more engaging, interactive and fun” that counteracts the drop in enjoyment interest and self-efficacy as they progress through the course (Teague & Roe 2008; Zingaro 2015). Collaborative problem solving has been found to increase student confidence, participation and student ability (Falkner & Munro 2009). However, one study on how students interact with one another during collaborative programming tasks found that some learning principles may be absent from student-student interactions such as deep questioning, especially with online learning (Gaspar et al. 2016).

One widely adopted peer learning strategies is pair programming. Pair programming originated from Agile methodologies and involves two software developers working on one piece of code together: a driver and a navigator (Teague & Roe 2008; Williams et al. 2002). A number of studies have applied the practice in the classroom. The benefits of pair programming used in education include (Hanks et al. 2011; Salleh, Mendes & Grundy 2014; Teague & Roe 2008; Williams et al. 2002):

- Improved results, satisfaction and enjoyment in introductory courses
- Some evidence that women, in particular, benefit from pair programming (Hanks et al. 2011)
- Improved self-sufficiency and less reliance on teaching staff
- Paired students demonstrate higher order thinking skills than those who work alone
However some studies have reported challenges in the use of pair programming such as concern for scheduling and partner compatibility (Hanks et al. 2011). Also, the skill level of students impacts the effectiveness of pair programming (Salleh, Mendes & Grundy 2011) - “the most confident students liked pairing the most, while the least confident students liked it the least” (Hanks 2006).

Student mentoring is another strategy that educators have used to try to help novice programmers. Student mentoring services have been found to assist students in building up their confidence in programming, but at the same time, student mentors gain confidence, experience and improve their communication skills (D’Souza et al. 2008). While the mentoring services are aimed at students that are struggling or failing, it has been found that often the students attending the mentoring service are those looking to obtain higher grades (Devey & Carbone 2011).

### 2.5.2 Student Social Networks

While some students prefer to work alone, many students do their learning within a social network of some form, learning from and obtaining help from various people. Every higher education teacher has probably heard a student mention they got help from the proverbial “friend”.

With the recent proliferation of social media, student use of social networking sites has become ubiquitous. Students have any number of platforms available to them to connect with each other, such as Facebook, mobile apps, wikis, file sharing etc (Liccardi et al. 2007). While the vast majority of students use Facebook, many see it as a platform for socialising rather than for education (Cheung, Chiu & Lee 2011; Tian et al. 2011).

Educators have looked to use platforms such as Facebook as part of their teaching, and this is discussed further in section 2.5.4. Students form friendships and support groups that extend beyond the classroom, and this now also happens via social media platforms. There have been limited studies, however, investigating the social networks programming students make use of and how they use them.

A recent study by Tsai (2012) has investigated the information behaviour of higher education students and looked at who they seek information from. It was found that when students are seeking information with course related problems, they are most likely to consult peers, teaching assistants and professors. They will occasionally consult parents. When seeking moral support, they are most likely to consult their family members, friends, peers and occasionally teaching staff (Tsai 2010a, 2010b; Tsai & Kim 2013).
One study by Sheard et al. (2013) found that few programming students sought help from teaching staff, with those who did usually consulting a help desk tutor. Most students indicated that they would ask their friends online but this was not to be depended on as friends were likely to be facing the same problems. Few students described discussing their work with others (Sheard et al. 2013).

More understanding is needed of how programming students use their social networks, both those provided by their educators and also those they create amongst themselves.

### 2.5.3 Information Seeking

Programming students have a wide range of information sources at their disposal. Generally, students are provided course material such as lecture notes, tutorial exercises and a textbook to work from. There are also programming resources available online that students can search for and read, but many of these are intended for professional programmers. While many teachers may assume that their students primarily use the information resources provided to them, a number of studies have investigated how programming students actually seek and use information.

Edwards (2005) identified four conceptions of information search that university students, in general, have when seeking information from the Web (Edwards 2005; Edwards & Bruce 2002, 2006).

- **Looking for a needle in a haystack.** They are focussed only on the topic they are searching for, without any planning or reflection on the search itself. They may not understand what they find or recognise the answer if they see it.

- **Finding a way through a maze.** The focus is on the search process itself as much as the topic being sought. There is effort put into planning and executing the search terms and sources used.

- **Using the tools as a filter.** They see searching as using the tools to sift through the junk to provide a usable smaller set of results. They don’t need to understand the topic to conduct the search and the tools can be used to further enhance their understanding of the topic.

- **Panning for gold.** Using the tools to limit the results to high quality resources during the search process. They focus on using the right tools to find the right resources.
More specific to programming, Postner and Stevens (2005) compared programming student information seeking activity between lectures and a course provided newsgroup. They observed students in lectures and noted that 30% took notes and asked questions, 45% just listened to the lecture and 25% did not pay attention and were doing some other activity. In analysing the newsgroup postings, they found that while 30% of students posted on the newsgroup, only 5% of students posted 10 or more messages. Some students actively read the newsgroup but never posted. They found that the newsgroup offered a very different form opportunity to ask questions than the lecture. Students were posting to the newsgroup while they were programming – at night and over the weekend (Postner & Stevens 2005).

One study investigated the preferred information sources of programming students. They found that students gave more importance to online help, product help (help system) and asking their peers. They showed a lower preference for going to classes and using e-learning systems (Costa, Aparicio & Pierce 2009).

Another study that also looked at the preferred information sources of programming students was by Blaho et al. (2013). They found that their students preferred online sources but their use of information sources was different between doing practical work and exam preparation. Table 2-4 summarises these findings. They also reported that only half of their students are interested in finding code that is functional for their needs and are not interested in the quality of the code.

<table>
<thead>
<tr>
<th>Information Source</th>
<th>Percentage of Student Use for Practical Work</th>
<th>Percentage of Student Use for Exam</th>
</tr>
</thead>
<tbody>
<tr>
<td>Online Help</td>
<td>80%</td>
<td>40%</td>
</tr>
<tr>
<td>Product Help</td>
<td>30%</td>
<td>20%</td>
</tr>
<tr>
<td>Books</td>
<td>25%</td>
<td>40%</td>
</tr>
<tr>
<td>Lectures</td>
<td>70%</td>
<td>90%</td>
</tr>
<tr>
<td>Forum</td>
<td>80%</td>
<td>50%</td>
</tr>
<tr>
<td>Peers</td>
<td>70%</td>
<td>70%</td>
</tr>
</tbody>
</table>

Table 2-4 - Programming Student Preferred Information Sources (Blaho et al. 2013)
A study by Sheard et al. (2013) found that the Internet and Google appeared to be the main source of help and the first port of call, as a source of alternative explanations or to obtain sample code. Few students reported reading the textbook. Instead they used it as a reference when they got stuck, to clarify a concept or as a source of sample code (Sheard et al. 2013).

A recent investigation into how programming students use the Web was conducted by Shi (2016). The vast majority of their students preferred to use the Web because they thought it was convenient (90%) and the best place to find information (67%). Some students (20%) preferred using the Web to avoid asking “silly” questions and appearing stupid in front of their lecturers and peers. The majority (60%) of students used the Web to further investigate concepts found in the lectures, learn new concepts and when struggling with a problem. Only 40% of students used the Web for revision (Shi 2016). They also found that most students preferred live programming demonstrations as a rapid way of learning programming, such as in-class demonstrations or online videos (Shi & White 2013).

### 2.5.4 Online Learning Forums

Learning Management Systems (LMS) are now in widespread use in universities. These often include discussion forums that are provided for students and tutors to ask and answer questions about their learning activities (Sheard et al. 2013). More recently, Social Networking Sites have become ubiquitous (Tess 2013) with almost all university students now having Facebook profiles (Davis III et al. 2012). With a large number of students using social networking sites (Facebook, Twitter, Myspace, LinkedIn and Blogs) educators have looked at incorporating such services into their teaching as an alternative discussion forum. On the whole most experiences have been positive but some reports do have reservations (Davis III et al. 2012; Liccardi et al. 2007; Tess 2013).

One study analysed their student online forum activity using Kolb’s learning style inventory and contrasted them with four online forum participation types: ‘Replier’, ‘Asker’, ‘Watcher’, and ‘No activity’. They found that Kolb’s ‘Accommodator’ style was associated with superior learning and that the ‘Replier’ participation type was also associated with higher scores (Shaw 2012). Use of online discussion forums was found to be more beneficial to small groups who were more active than the larger groups (Shaw 2013). Sharma, Shen and Goodwin (2016) suggest that student participation in online forums within an LMS can provide an indicator to teachers about the level of engagement of the students.
A number of studies have investigated using Facebook groups to support their programming students. Pieterse and Rooyen (2011) provided students with a Facebook group in addition to their formal discussion forum. They found that the activity on the formal forum was rather impersonal while the discussions on the Facebook group seemed to create more of a student community (Pieterse & Rooyen 2011). In a study that used Facebook Groups to motivate novice to novice programming student interactions found that interactions led to social learning. Facebook groups also encouraged engagement and the students to support each other (Maleko et al. 2014; Maleko et al. 2013). They also found that while Facebook attracted more students, Blackboard was viewed as the authoritative medium for official course material (Maleko et al. 2013).

Hundhausen and Carter (2014) created Facebook groups for two programming units. They found that the majority of students actively used the Facebook group, with the students doing the more advanced unit being more active (80%). In contrast, Sharma, Shen and Goodwin (2016) found that students were less active in the course Facebook group compared to the online discussion forum provided through the LMS.

In a study that investigated the use of a Mobile Social Learning Environment (MSLE) to support novice-to-novice interactions, it was found that the increased interactions among the novices enabled them to form learning communities where they shared their knowledge and ideas about the programming language as well as providing them with instant access to discussions and feedback “anytime and anywhere”. “Some novices were able to provide assistance to others, hence indirectly, strengthening their own knowledge of programming” (Maleko 2014; Maleko, Hamilton & D'Souza 2012).

A recent study by Hundhausen, Carter and Adesope (2015) provided students with two types of discussion forums. One was within the learning management system, while the other one was a plug-in integrated directly into the student programming environment (Visual Studio). This allowed the students to read and post questions to each other while they were writing the code. They found that while use of both forums was positively correlated with student grades, the students who used the plug-in were twice as active. The type of activities they noted students performing within the forum were (Carter, AS & Hundhausen 2015):

- Observing others’ activities and progress
- Observing others’ problem solving processes
- Observing other’s social interactions
- Asking questions of others
- Answering other’s questions
Further analysis of the plug-in logs revealed that three-quarters of the questions related to language or implementation and that helpful answers to questions posed were positively correlated with students' ability to make progress with their code (Carter, AS & Hundhausen 2016).

2.5.5 Public Programming Discussion Forums

Online discussion forums have increasingly become large and popular repositories of problem solving-solutions for programming. A number of studies have investigated the information seeking behaviours of programming students using the Stack Overflow discussion forum, which is perceived by students as a popular and useful forum (Eickhoff et al. 2014; Lu & Hsiao 2016). Novice and experienced programming students have been found to have different information seeking behaviours (Lu & Hsiao 2016):

- Students actively search and read programming related posts following course schedule topics.
- Advanced students refine their queries, examine search results and read them.
- Novice students do not refine their queries and only skim through the posts.

Novice students using Stack Overflow often required more assistance in formulating search queries and filtering relevant information and in understanding basic programming concepts (Lu, Hsiao & Li 2016). They identified four categories of student information seeking behaviours: hyper-user, selector, impatient-reader and passive-user. They found that the majority of students were passive-users, mostly novices. Further investigation of these passive-proactive learners (known as lurkers (Chi & Wylie 2014)) suggest that learning still takes place when novices read the postings and replies of others. The process of reading all the answers to a posting was suggested to help with classifying and discerning relevant information which in turn helps with learning (Hsiao 2015a, 2015b; Hsiao & Naveed 2015).

2.5.6 Summary

While there has been considerable research into how students collaborate in class, there has been limited research into how programming students use their social networks, how they seek information and what information sources they use and how they use online discussion forums. As much of the learning now happens outside the classroom, more understanding is needed of these student activities.
2.6 Learner Categorisations

There have been a number of previous studies that have developed categorisations of learners and learning strategies, in order to better understand the learning experience of programming and gain a deeper understanding of the students themselves.

A widely cited categorisation of the general continuum of stages from novice to expert is that by Dreyfus, Dreyfus and Athanasiou (2000):

- **Novice** – Learns facts, features and rules.
- **Advanced Beginner** – Starts to recognize and handle situations with quite understanding.
- **Competence** – Considers the whole situation and chooses an organised plan to achieve the goal.
- **Proficiency** – No longer has to consciously reason through all the steps to determine a plan.
- **Expert** – Knows what to do based on practiced understanding.

**Stoppers and Movers (Perkins Et Al)**

Some early studies that discussed student programming categorisations were that of Perkins et al. (1989), where the researchers sat with the participants observing them as they attempted to complete a programming task. During these sessions, students were asked how and why they did certain tasks. The result of this study distinguishes two main kinds of novices in terms of how they deal with programming problems.

- **Stoppers**
  Stoppers are those students who abandon all hope of solving a problem on their own when confronted with a problem or lack of clear direction to follow. Students’ attitudes to errors or mistakes are important. Those who become frustrated or have a negative emotional reaction to errors are likely to become stoppers.

- **Movers**
  Movers are those students who keep trying, experimenting and modifying their code. They use feedback about errors effectively and are likely to solve their current problem and progress.

- **Extreme Movers (tinkerers)**
  Extreme movers are those students who are not able to trace/track their program and tend to make random changes. Similar to stoppers, they have little chance to progress.
**Rocket Scientists and Strugglers (Jenkins and Davy)**

In their paper, Jenkins and Davy (2000) describe how they divided their student cohorts into four distinct groups, which they then treated differently in relation to teaching, to address their specific level of programming knowledge. These groups were:

- **Rocket Scientists**
  Students who were already highly proficient programmers and who would learn little from the module.

- **Averages**
  Students who didn’t fit any other category. They would be expected to pass the module well and need only the occasional word of advice or help with debugging.

- **Strugglers**
  Students who would find the course challenging but be expected to pass reasonably well.

- **Serious Strugglers**
  Students who would find the course extremely difficult and not be expected to pass without significant additional support and encouragement.

**Effective and Ineffective Novices (Robins, Rountree and Rountree)**

Robins, A, Rountree and Rountree (2003) present a thorough review of the literature relating to the psychological/educational study of programming. They identify general trends comparing Novice programming characteristics versus Expert programming characteristics focusing primarily on the deficits of novices. They suggest two key distinctive types of novices that of effective and ineffective novices.

- **Effective Novices**
  Those students who learn to program without excessive effort or assistance.

- **Ineffective Novices**
  Those students who do not learn to program, or do so only after inordinate effort and personal attention.

They further suggest that the most significant differences between effective and ineffective novices relates to the strategies they employ in accessing and applying the knowledge about the language to comprehend and generate programs rather than the knowledge itself.
Piaget Stages of Programming Maturity (Lister and Teague)

Lister and Teague developed a categorisation of programming students based on neo-Piagetian theory (Lister 2011, 2016; Teague 2015; Teague et al. 2013; Teague et al. 2012; Teague & Lister 2014a, 2014b; Teague, Lister & Ahadi 2015). The four-stage model of learning to program reflects Piaget’s developmental stages from least mature to most mature:

- **Sensorimotor (Pre-tracing stage):** The novice programmer has an incoherent understanding of how the program works. They have difficulty tracing code and may swap between conceptions (right or wrong) depending on the code they are viewing at the time. They tend to learn by the act of tracing the program themselves and then receiving feedback.
- **Preoperational (Tracing stage):** The novice can manually execute (trace) multiple lines of code and make inductive guesses about what the code does. They do so by performing one or more traces and looking at the relationship between inputs and outputs.
- **Concrete Operational (Post-Tracing / Abstract Tracing Stage):** The novice can reason about the code deductively by reading the code itself. At this stage students start to show a purposeful approach to writing code.
- **Formal Operational:** This is the level at which expert programmers perform. They can reason logically, consistently and systematically about the code. This level also requires reflective capacity (thinking about how they think).

Ways of Experiencing the Act of Learning to Program (Bruce Et Al)

The study by Bruce et al. (2004), sought to understand the act of learning introductory programming from the perspective of first year university IT students. These perspectives are described in a set of five different categories or ways students experience the act of learning programming:

**Category 1: Following:**

The ‘Following’ category describes how learning to program is experienced as following the structure of the unit to get through it (pass the course). These students struggle to keep up with tasks and focus primarily on completing the tasks that gain marks, such as assignments. They seek feedback from teaching staff to check if they are on track. They experience frustration if the course materials are presented in a way that does not match their expectations or perceived needs (i.e. such as the material being presented at a fast pace).
Category 2: Coding:

Learning to program is experienced as learning to code. Learning the syntax of the programming language is important, in order to learn to code and to learn to program. They persevere in trying to work out syntax. They seek extra guidance from teaching staff for specific examples of code or solutions as time spent discovering their own solutions is considered a waste of time rather than a process of discovery. Sometimes they seek sample pieces of code from texts, the internet or from other sources to try out using trial and error when entering or typing the code they want to adopt. During this process, they expect intensive direction from experts and teachers experiencing frustration and dissatisfaction if it is not provided. They also use elements of rote learning.

Category 3: Understanding and Integrating:

This category of students sees understanding as an integral aspect to learning how to program. They seek understanding of the big picture, what they have done in order to get the intended outcome. Their learning motivation is a desire for insight – understanding, not just completing a task. Students gain understanding of concepts by doing the task where they see it as involving a progressive sequence of concepts, building on prior experience. They use a “building Block” approach to understanding and writing programs trying to understand prior moving on to the next concept. They persevere implementing the same thing in different ways. They play and experiment with code which helps them integrate prior experience and develop understanding of concepts. These students seek variation in their sources of explanations of concepts. For these students, becoming more proficient with coding is the turning point where their focus shifts from solely coding to understanding.

Being able to program means being able to apply concepts in different ways, to use what has been done in the past in new contexts. Frustration my happen if they feel they are not keeping up with understanding- They may show dissatisfaction with course structure.

Category 4: Problem Solving:

Learning to program is experienced as learning to do what it takes to solve problems. The student learning motivation is the problem the student is attempting to solve. They approach understanding by looking at the problem and the program as part of the broader context. The task to be solved provides the motivation to learn how to write the program and also the means to achieve understanding. The primary focus is solving the problem and coding is seen as part of the learning process. Planning is seeing as important prior actually typing in the code and there may be a tension between desire to solve the problem and jumping straight into coding. Learning the language is not a means in itself.
Category 5: Participating or Enculturation

Learning to program is experienced as learning what it takes to be part of the programming community who share a language and culture. This involves understanding what the programmer thinks as well as understanding what the programmer does and using programs to communicate with other programmers. Communicating with other programmers is a strategy used by the students who see learning programming in this way. For example, readability of a program is considered an important feature of a good program. To share in the culture of programming, the programming language needs to be understood in the context that it is being used.

Challenging Nature of Programming (Dunican)

A Grounded Theory study by Dunican (2006) presents the Challenging Nature of Programming Theory which describes the major phenomena experienced by first year programming students within the classroom (see Figure 2-5). The study interviewed students from four Irish higher education institutes. The theory developed was used to create a taxonomy of novice computer programming learner types, presented in the form of a learning continuum (see Figure 2-6) (Dunican 2006):

Figure 2-5 - Challenging Nature of Programming Theory (Dunican 2006)
The programming learner types described by Dunican (2006) were:

- **Disengager**
  The disengager sees the challenging nature of programming in a negative light. They are likely to find the experience overwhelming and find the material confusing for the whole duration of their course. Their state of mind is that of bewilderment, feeling totally lost and isolated. They experience constant self-doubt and feelings of being stuck. They display a negative self-efficacy, where they attribute learning to ability and not effort. They have a negative perception of programming, see it as being too difficult and find the material alien. They are unlikely to have prior programming knowledge. They find the material delivery too fast and too deep and find it difficult to make sense of the material, even for basic concepts. With their dealing strategies, they persevere on a random basis. They seek help from their peers, usually looking for a full solution. They may practice further avoidance strategies such as skipping over the material, and eventually disengage, become disinterested and withdrawing.

- **Slogger**
  The slogger finds the challenging nature of programming a predominantly negative experience but persists in trudging through it, with some minor success. They find a good part of the material difficult to grasp and their state of mind is mostly that of confusion and bewilderment and self-doubt. They also have a negative awareness of others because they compare the amount of work they need to do with their peers. They have a negative perception of programming, finding it hard and difficult to grasp and thus intimidating. They find new concepts in the material alien and difficult to understand.
They find the pace of delivery challenging, often moving on to subsequent topics without having understood the previous ones. They look for coding examples that can help clarify concepts but if these are ineffective, they end up struggling and not fully understanding. They practice progression strategies in that they persist to try to understand the material by looking at various sources repeatedly. They also skip over the material until a full explanation is received from the lecturer. They seek similar assistance from their peers. They randomly guess changes in code without any strategy. They disengage with parts of the syllabus that they cannot understand or when they cannot receive full assistance from peers or lecturers.

Worker

The worker student types alternate between positive and negative attitudes towards the challenging nature of programming. When topics progress and become more difficult, they find it very frustrating (view programming as intimidating and daunting) but they work hard at it until they succeed to some extent and then they end up finding it rewarding (viewing programming as interesting and enjoyable). This positive attitude only lasts until they encounter the next level of difficulty where the bar has been raised drifting back to the negative zone. In the negative zone their state of mind is that of confusion, bewilderment and self-doubt with some level of understanding. Unlike the earlier learner types, they have a desire to achieve or accomplish which helps them move out from the negative zone. In their awareness of others they are conscious of how others experience similar difficulties and see this as an incentive to continue working on it.

Their perception of programming is less negative than the disengager and slogger but they still find the more complex material alien and difficult which prompts them to work harder by repeating it and doing more examples to find different ways of doing things. In some cases, hard work will not help them to understand the more complex topics, in which case they accept it and move on. In other cases, some topics will take much longer for them to understand. They seek full or partial assistance from either peers or lecturers. With peers this can be looking at what others did or helping each other. They practice planned guessing where they remove parts they don’t understand and use trial and error to see what it does and they work through tasks bit by bit. They may skip over material until solutions are provided with the aim to understand where they went wrong. While in the negative zone they may disengage by postponing working on it.
Manager

The manager appears to be able to cope or manage better the challenging demands of programming. They have a primarily positive perception of programming (enjoyable and interesting) where they may find some topics alien and difficult at the start but doable. Even though they may not know all the material, they attain a good solid base of programming. Their state of mind indicates that they may experience some level of confusion and self-doubt finding programming challenging but work through it diligently and they don’t experience a great level of pressure. They also are better able to deal with confusing material compared with the strugglers as they are more comfortable with programming. They are also more accepting of syntax nuances and willing to tackle intricate ways of coding. Their awareness of others struggling does not impact on them but other’s achievements serve as motivators to keep going.

With challenging material such as object orientation, they work with various examples that gradually build up in order to understand. They also ask questions prior moving on to other topics. They are comfortable not understanding concepts in the short term knowing that if they work through them, they will understand eventually. They mostly use progression strategies but tend to skip over topics they considered of less critical importance. Their persistence in working through multiple examples helps them build up their skills and confidence. They engage in peer learning where they collectively analyse a project to check if everyone in the group is on the right track and also they bounce ideas off each other. They use a more planned approach where they may break a task into manageable chunks or are able to transfer bits of code from simpler code into methods or functions. The re-implementation of code chunks they practice helps them build a library of know bits of code.

Achiever

The achiever has a high level of competence in programming and progresses without experiencing regular difficulty with programming concepts. They have a positive perception of programming but may on occasion become bored with the material or find some material topic alien if they haven’t learnt it before. They have a confident state of mind and are unlikely to experience confusion or bewilderment. When they find the pace of delivery and depth of material non-challenging they may work on things that interest them on their own. They engage mostly in progression strategies persisting through any new difficulty and enjoy a sense of accomplishment when completing a challenging project. They are aware of other’s struggles but this does not impact on them.
Narratives on Learning to Program (Cardell-Oliver)

A recent study by Cardell-Oliver (2014) involved analysing various artefacts produced by students during teaching, including program code, assignments, course feedback surveys, emails, and comments written on examination papers. From these, she developed narratives to describe four student experiences of a first programming course:

- **Thriving**
  A thriving student is a student who thrives on the feedback provided. This student has a positive perception of their ability and effective strategies for overcoming problems when they are encountered. They have effective strategies for organising their knowledge. They are motivated by getting their programs to work and acquire and practice skills, and practice integrating those skills.

- **Surviving**
  These are performance-directed learners who focus on what has to be done to achieve good marks. Their learning is driven by the course assessment criteria. They are motivated by success and practice the component skills of programming but may have poor skills in organising knowledge. They are not aware of gaps in their knowledge structures and focus on optimising their performance in assessments.

- **Drowning**
  This is a student who is working hard and appears to be managing, but has actually failed to understand fundamental concepts. They are motivated and invest time and effort, but their learning is hindered by misinterpreting feedback. Their learning strategies tend to be shallow and they have poor organisation of their knowledge. These students may experience many distractions which may be unconscious avoidance strategies. They are also hampered by poor advice they receive from other students.

- **Lost**
  These are low-achieving learners who became lost early on and never catch up. They have the potential to succeed, but are hindered by their inability to identify how and when to respond to problems. They start with the motivation to succeed and have learning strategies that have served them well in the past, but can hinder their learning of programming. Their emotional responses when faced with difficulties (panic attacks and feeling helpless) prevent them from adjusting their learning strategies, which leads to poor results and then lost motivation.
2.7 Chapter Summary

This chapter presented a review of the literature relevant to this study, with the student perspective as the central theme. Topics included success factors, student motivations, learning behaviours, problem solving, social networks, information behaviour and learner categorisations. While there has been considerable research into how students learn programming, much of this previous research has focussed on what happens inside the classroom. More understanding is needed of what students do outside of the classroom.

While these previous studies provide valuable insights into the student learning experience, each individually sheds light on a small piece of the puzzle. A cohesive theory is needed that incorporates the wide range of influences on the programming learning experience and how they influence each other.

This goal of this study was to develop such a theory. Chapter 3 explains the research design and methodology that was used to develop the theory and then chapters 4, 5 and 6 present and describe the theory in detail.
Chapter 3  Research Design and Methodology

As highlighted in chapters 1 and 2, the goal of this study was to develop a cohesive theory that gives deeper insights into the influences on the programming learning experience, from the student’s perspective. The aim of qualitative research is “to achieve a deeper understanding of a phenomenon” (Kvale 1989). A qualitative research approach was therefore chosen for this study. Section 3.2 explains this choice in detail and section 3.3 presents the research plan followed by this study.

The research methodology used in this study was Grounded Theory (Strauss & Corbin 1998). Section 3.4 explains the fundamental concepts and activities of Grounded Theory. In order to investigate the learning experience of programming students in depth, thirty-one students were interviewed. Section 3.5 provides an overview of the students interviewed for this study, how they were interviewed and discusses the sample size.

Analysis of the interview transcripts, using Grounded Theory principles, produced concepts, categories and themes that evolved, over multiple iterations, into a cohesive theory that answers the research questions and is presented in this thesis. Section 3.6 describes this process, including a discussion of the software package used (Nvivo), theoretical sampling and how the research journey unfolded.

By following a rigorous research plan, the theory developed in this study represents the concepts found in the data. Section 3.7 discusses how reliability and validity were ensured.

3.1 Research Questions

This study sought to answer the following research question (introduced in Chapter 1):

*What are the key influences on the learning experience of programming students?*

Related subsidiary questions were:

*What influences motivate students to learn how to program?*

*What types of learners do we find when teaching programming?*

*How do students use their personal networks in learning programming?*

*How do students use information sources in learning programming?*
3.2 Research Approach

There are three main research perspectives presented by Myers (2009): positivist, critical and interpretive. Positivist researchers assume that reality is objective and can be described by measurable properties. Positivist studies generally attempt to test theory and increase the predictive understanding of phenomena. Critical researchers assume that social reality is constrained by various forms of social, cultural and political domination. The main task in critical research is one of social critique. Interpretive researchers assume that knowledge is gained through social constructs such as language and shared meanings. Interpretive research focuses on the complexity of human sense making and attempts to understand phenomena through the meanings that people assign to them (Myers 2009; Walsham 1995). The interpretive approach is well established in Information Systems research (Walsham 2006).

The research perspective and methodology employed in a study should be determined by the nature of the research question (Creswell 2009). The research question of this study was to explore and understand the learning experience of programming students. The goal of the research was to generate theory, not to test theory. There was no cultural or political agenda involved. Therefore, this study took an interpretive research perspective as being the most appropriate to answer the research question.

The interpretive research in this study was guided by the seven principles for interpretive field studies described by Klein and Myers (1999). These are:

- **The Fundamental Principle of the Hermeneutic Circle.**  
  This principle suggests that all human understanding is achieved by iterating between the parts and the whole that they form. The methodology used in this study, Grounded Theory, is inherently iterative in nature.

- **The Principle of Contextualization.**  
  This principle requires critical reflection of the wider social and historical context of the area being investigated. This study sought to understand the programming learning experience in the context of the wider learning environment, both inside and outside the classroom.

- **The Principle of Interaction Between the Researchers and the Subjects.**  
  This principle requires critical reflection on how the data was obtained through the interaction between the researchers and the participants. The researcher in this study, being a programming teacher, needed to be aware of potential influences and biases when conducting interviews with students and analysing their responses.
The Principle of Abstraction and Generalisation.

The concepts that are revealed through the interpretation of the data in detail should be abstracted to a higher theoretical level. The goal of this research was to develop a theory at a higher abstract level that not only provides deeper understanding of programming students but may potentially be applicable in other learning contexts.

The Principle of Dialogical Reasoning.

This principle requires the researcher to be sensitive to contradictions between the literature and the actual findings of the study. In Grounded Theory, the theory emerges from the data and is not influenced by previous theories. Chapter 7 provides a comparison between the theory developed in this study and existing theories.

The Principle of Multiple Interpretations.

This requires sensitivity to possible differences in interpretations among the participants. By interviewing a wide range of students with different experiences, this study obtained a range of interpretations of the learning process.

The Principle of Suspicion.

This requires sensitivity to possible biases or distortions in the transcripts collected from interviewees. The interview questions were open ended and designed to allow the students to express their own experiences and opinions freely, in a comfortable manner.

Qualitative research is “a nonmathematical process of interpretation, carried out for the purpose of discovering concepts and relationships in raw data and then organizing these into a theoretical explanatory scheme” (Strauss & Corbin 1998). Reasons for doing qualitative research include understanding the meaning or experience of people and obtaining intricate details about phenomena, such as feelings, thought processes and emotions (Strauss & Corbin 1998).

The research question of this study sought to investigate the influences on the learning experience of learning to program. This required asking students what it is that they do and about their feelings, thoughts and perceptions. The goal was to build a theory that gives deeper insights into the behaviours and influencing factors. Therefore, a qualitative research approach was appropriate for this study.
3.3 Research Plan

This study used a research plan developed by Bruno (2011), which is itself based on the theory building process described in Eisenhardt (1989) and augmented with steps from Grounded Theory – open, axial and selective coding (Strauss & Corbin 1998). The plan includes multiple iterative loops (see Figure 3-1).

![Figure 3-1 - Research Plan: theory building process (Eisenhardt 1989) combined with Grounded Theory (Strauss & Corbin 1998) (Bruno (2011))](image)

The research plan includes the following steps:

1) Getting Started.

An initial definition of a research question is needed (at least in broad terms) and early constructs may also be useful. Both are tentative, however, as the constructs will evolve during analysis and the research question may be refocussed as the findings develop.
2) **Selecting cases.**
Defines the population from which participants will be drawn. Eisenhardt promotes theoretical sampling, as do Strauss and Corbin (1998). Cases are chosen for their potential contribution to the developing theory. This study sought to interview a range of Information Systems students.

3) **Crafting Instruments.**
This study performed interviews that included open questions about the student programming experience to generate rich data.

4) **Entering the field.**
This involves not only gathering data but also the initial coding and analysis in an overlapping, iterative process. In the early stages of this study, interviews were transcribed and analysed as they occurred to enable emerging concepts to inform the following interviews.

5) **Analysing data**
Analysing the data is an important stage in building theory. Eisenhardt promotes not only within-case data analysis (analysing each case in detail) but also cross-case data analysis (looking for similarities and differences between cases).

6) **Shaping hypotheses**
Tentative themes, concepts, and relationships begin to emerge. In shaping the hypotheses, a highly iterative process is used to constantly compare the theory with the data – iterating toward a theory which closely fits the data and the represents the cases.

7) **Enfolding Literature**
The emergent concepts and theory are compared to those in the literature for support or contradictions. This enhances the internal validity and generalisability of the theory. Enfolding the literature is important, because the number of interviews are limited.

8) **Reaching Closure** involves deciding when to stop adding interviews and doing analysis. Usually this is at the point of theoretical saturation, when new cases do not add to the emerging theory, but considerations such as time and money may also dictate when this occurs.

The iterative loops are key elements to the research plan (shown with dotted ellipses in Figure 3-1). The first iteration (a) occurs during data collection, with interviews being transcribed and analysed. The second iteration (b) shows the hypotheses developing (using axial coding) as further interviews are conducted and analysed. The third iteration (c) describes the shaping of the hypothesis towards a theory (using selective coding).
3.4 Methodology

Myers (2009) describes four research methods appropriate for qualitative research: action research, ethnography, case studies and Grounded Theory. Action research involves the researcher actioning change in an organisation and studying the effect of those changes (Myers 2009). As this study sought to understand existing student behaviours, action research was not appropriate. Ethnographic research involves studying a group in a natural setting Creswell (2009). While it would have been possible to observe student activity in the classroom, this would only have revealed how students performed classroom tasks. This study sought to investigate student behaviour both inside and outside the classroom. Case study research involves studying people and phenomenon within a real-life organisation (Myers 2009; Yin 2009). This study sought to investigate individual student behaviours and there was no organisation to study.

3.4.1 Grounded Theory

The methodology most appropriate to answer the research questions in this study was Grounded Theory. A grounded theory denotes a set of well-developed categories, themes and concepts that are inter-related through statements of relationship to form a theoretical framework that explains a social, psychological, educational or other phenomenon (Strauss & Corbin 1998).

“Grounded theory begins at the level of observation and concludes at the conceptual level. Thus, concepts or variables are created by observation of the data. Data is gathered within a reflective framework where a broad research question is raised. Interviews are conducted, the data or recorded statements from the interviews are analysed, and a second set of interviews is conducted. After that, further analysis occurs with reflection and the formulation of provisional hypotheses.” (McMurray, Pace & Scott 2004).

Grounded theory was originally developed by Glaser and Strauss (1967) and has become widely used in social research and other fields including Information Systems. Over the years, Grounded Theory has developed into a number of strains, but they all share the same common elements and the goal of generating a theory that is grounded in the data. This study conducted Grounded Theory as described by Strauss and Corbin (1998).
The primary analysis method of Grounded Theory is the coding of data into concepts and themes that are then developed into a theory. There are three different forms of coding: open coding, axial coding and selective coding. Figure 3-2 shows a pictorial view of how raw data is broken up into concepts, refined into themes and then reconstructed into categories and eventually a theory (Bruno 2011).

Two fundamental concepts of Grounded Theory are objectivity and sensitivity. Objectivity means maintaining an objective stance when analysing the data - “a willingness to listen and to ‘give voice’ to respondents” (Strauss & Corbin 1998). Sensitivity means “having insight into, and being able to give meaning to, the events and happenings in data. It means being able to see beneath the obvious to discover the new” (Strauss & Corbin 1998).

A common myth of Grounded Theory is that it requires a researcher to start analysis with no prior knowledge or reference to the literature. “The idea that reasonable research can be conducted without a clear research question and absent theory simply defies logic” (Suddaby 2006). Existing knowledge is not put aside, but it is acknowledged and understood.
Existing literature can be used to develop the research question and interview questions, but the questions must be open to not lead the respondents. During analysis, concepts and theories from the literature can be used to make comparisons with concepts arising out of the data. The developed theory is compared with theories from the literature to confirm findings and show where the literature is incorrect or only partially explains the problem (Eisenhardt 1989; Strauss & Corbin 1998).

The basic operations in Grounded Theory analysis are those of constantly asking questions and making comparisons. Strauss and Corbin (1998) describe many forms of questions, including who, when, why, where and what. Other forms of questions are spatial, technological, informational and cultural. The constant making of comparisons includes (Strauss & Corbin 1998):

- Cross-case comparisons. Each case is compared to other cases at the property or dimensional level for similarities and difference. Comparisons were constantly made between students in the study to reveal both common and contrasting learning experiences, the way they used learning resources and collaborated with their peers.

- Theoretical comparisons. This may involve comparing an incident to something else at a conceptual level, which could be from the literature or from experience. For example, during the analysis, various behaviours were compared to the fixed and growth mindset concepts developed by Dwek (2008).

- The Flip-Flop Technique involves taking a concept and turning it ‘inside out’ to obtain a different perspective. Looking at opposites or extremes may bring out significant properties. An example of this in the analysis was investigating and contrasting the behaviours at each extreme of the spectrum – students that were totally engrossed in programming and students that were totally disinterested in learning it.

Memos are written records made during analysis and are an integral part of the Grounded Theory methodology. Memos can be coding notes, which contain the products of the three types of coding (open, axial and selective), theoretical notes, which record thoughts and ideas during the analysis process, or operational notes. Memos record the progress, thoughts, feelings and directions of the research and researcher (Strauss & Corbin 1998).
3.4.2 Coding

A major task in Grounded Theory is coding the data into concepts, categories and themes. Incidents, ideas, events and acts in the data are broken down and named. The name may be given by the analyst or taken from the data itself, known as *in-vivo* codes (Glaser & Strauss 1967).

Open coding aims to uncover and develop concepts from the data. Data is “broken down into discrete parts, closely examined and compared for similarities and differences” (Strauss & Corbin 1998). Initial analysis involves ‘Microscopic Examination of Data’ - “the detailed line-by-line analysis necessary at the beginning of a study to generate initial categories (with their concepts and themes)” (Strauss & Corbin 1998). As the analysis progresses, similar concepts are grouped together to form categories which are more abstract higher order concepts. Categories are then developed in terms of properties and dimensions. Properties are the characteristics of a category while dimensions represent the continuum or range of the property (Strauss & Corbin 1998). For example, some of the early concepts that arose from the analysis were intrinsic and extrinsic motivations, changes in motivation, career goals and interest, which were grouped into the ‘motivations’ category. These concepts arose early in the analysis of the first 7 interviews and then were sought in all the following interviews.

Axial coding involves reassembling the data that was fractured during open coding. This includes laying out categories in terms of their properties and dimensions, relating categories to their subcategories, finding variations in actions/consequences and looking for how major categories relate to each other (Strauss 1987). “Concepts that reach the status of a category are abstractions...they should have a relevance for, and be applicable to, all cases in the study” (Strauss & Corbin 1998). In practice both open and axial coding occur at the same time. Over multiple iterations, the major categories are integrated to form a larger theoretical scheme and the findings take the form of a theory (Strauss & Corbin 1998).

The motivations category described in the previous paragraph was further developed by comparing and contrasting it with dependency and usefulness of online sources. This led to further development of the motivations category by combining the intrinsic and extrinsic values with dependency and skill level. These groupings eventually developed into a higher abstracted category called Learner Nature.
Selective coding involves deciding on a central category, which “represents the main theme of the research... all the products of analysis condensed into a few words to explain ‘what this research is all about’ ” (Strauss & Corbin 1998). All the other categories and themes are integrated around the central category to form a cohesive theory. As the central category is refined and integrated with other concepts, the theory grows in depth and explanatory power (Strauss & Corbin 1998).

Selective coding in this study occurred with the development of Perceived Personal Relevance, Learning Trait and Skill Level, which focussed the analysis on the learning experience and led to the development of the Programming Learner Profiles as the central theme.

3.5 Data Collection

A range of data collection techniques are possible in qualitative research: interviews, participant observation and using documents (Myers 2009). This study sought to understand the learning experience of programming students in depth, from the student perspective.

Analysing documents such as student assignments and exams could have been possible, but these only represent the final outcome of the student learning process and not the learning process itself. Analysing posts on student forums would only give insights into the students who post, potentially leaving out a large section of the student population. Classroom observation of students was specifically ruled out by the ethics committee so was not possible. Even if it had been possible, observing students in their classroom activities would only have captured a part of their learning processes.

Interviews allowed open ended questions to be asked and for the students to describe, in detail, the activities they performed, both inside and outside the classroom. It also allowed for questions relating to the student’s perceptions, goals and expectations. Questions could also be varied in each interview to gain richer data into the activities described by each student.
3.5.1 Student Recruitment

A total of 31 students were interviewed for this study. Students were recruited from those that responded to flyers that were handed out during classes and seminars. Students were given a gift voucher as compensation for their time. The goal was to interview as wide a range of students as possible. The conditions of the ethics approval, however, prevented the researcher from selecting students based on background or grades due to privacy issues.

Nine of the students interviewed were female and 22 were male. This percentage of females (29%) is representative of the gender ratio in the Information Systems degree most students were drawn from and is also representative of the fact that 28% of ICT workers in Australia are female (ACS 2017).

Table 3-1 displays the age groups of the students interviewed. The majority of the students were in their early 20s, which is to be expected of undergraduate students. The majority of students were local Australian students (28) while 3 were international students.

<table>
<thead>
<tr>
<th>Age Group</th>
<th>Participants</th>
</tr>
</thead>
<tbody>
<tr>
<td>18-21</td>
<td>16</td>
</tr>
<tr>
<td>22-29</td>
<td>11</td>
</tr>
<tr>
<td>30-39</td>
<td>3</td>
</tr>
<tr>
<td>40+</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3-1 - Student Age Groups

Twenty-one students were studying full time, 3 part time and 7 were graduates, from 2 to 12 years since graduation. Table 3-2 lists the stage of the degree the students were in when interviewed.

<table>
<thead>
<tr>
<th>Stage in Degree</th>
<th>Participants</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Year</td>
<td>4</td>
</tr>
<tr>
<td>Second Year</td>
<td>10</td>
</tr>
<tr>
<td>Third Year</td>
<td>4</td>
</tr>
<tr>
<td>Final Year</td>
<td>6</td>
</tr>
<tr>
<td>Graduate</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 3-2 - Student Stage in Degree
While the majority of the students were studying Information Systems (or were Info Sys graduates), there were a number of students from other disciplines (see Table 3-3).

<table>
<thead>
<tr>
<th>Degree of Study</th>
<th>Participants</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information Systems</td>
<td>19</td>
</tr>
<tr>
<td>Accounting</td>
<td>2</td>
</tr>
<tr>
<td>Computer Science</td>
<td>1</td>
</tr>
<tr>
<td>Logistics</td>
<td>1</td>
</tr>
<tr>
<td>Marketing</td>
<td>1</td>
</tr>
<tr>
<td>Graduates (Info Sys)</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 3-3 - Student Degree of Study

3.5.2 Interviews

Interviews are a very important data gathering technique in qualitative research and allow the gathering of rich data (Myers 2009). Twenty-seven of the interviews in this study were conducted in person and 4 online using Skype (recording audio only). The interviews lasted between one to two hours and were recorded after obtaining consent from the students, following ethics requirements (see Appendix 9.3).

The questions were semi-structured but many of the questions were open ended so the students could discuss whatever they thought was relevant. This also made it possible for the interviewer to ask follow up questions as the interview unfolded. The students were asked to describe their experiences in learning programming. This included the activities they performed and how and when they performed them, but also how they felt about their experiences at the time. The questions tried to delve into their expectations and mindset at the time of their learning and how they dealt with difficulties. They were also asked when and how they sought help with problems. An example of the interview questions can be found in Appendix 9.1. The questions asked of the students evolved as the research progressed. The questions shown were the most recent version.

The first 10 interviews were transcribed by the primary researcher. This allowed the researcher to become immersed in the data and improve the analytical process. The remainder of the interviews were transcribed using a professional transcription service.
3.5.3 Sample Size

“An appropriate sample size for a qualitative study is one that adequately answers the research question” (Marshall 1996). Quantitative studies usually aim to test a theory on a given population so the findings can be generalized to the whole population. The sample size in such studies is determined statistically from the population being studied (Williamson & Johanson 2013).

As this was a Grounded Theory study, the goal was to generate a theory rather than test one. The number of participants in this case is determined by saturation of the theory, when further interviews only support the developed theory and do not give new insights (further described in section 3.6.2). As the analysis in this study required in-depth analysis of rich interview data, a smaller sample size was necessary to make the task manageable (Crouch & McKenzie 2006; Sandelowski 1986, 1995).

This study interviewed 31 students. This is comfortably in the range of 20 to 30 participants recommended for smaller qualitative projects (Charmaz 2006; Creswell 1998) and also close to the median sample size of 30 found in other Grounded Theory PhD studies (Mason 2010).

3.6 Data Analysis

This study analysed the data using the Grounded Theory methodology (Strauss & Corbin 1998). The key data analysis activities of Grounded Theory are described in section 3.4.1. This section gives a brief description of the software package used (Nvivo) and how some features of the tool were used. It also discusses theoretical sampling and how the analysis completed when saturation was reached. A reflection is made of the research journey this study took and how the analysis evolved, as well as a discussion about how potential researcher biases were acknowledged and dealt with.

3.6.1 Qualitative Research Tool - Nvivo

The software tool used to assist the analysis in this study was NVivo version 10 (QSR 2016). Such software is useful in the analytical process because it enables the researcher to store large amounts of data and easily sort, search and link them. It also enables the researcher to keep related documents, such as memos, together and link them with the data (Bazeley 2007).
The tool, however, does not do the analysis. “The intellectual work of actually conceptualizing can only be done by the brain of the researcher” (Webb 1999). The tool allows the data to be collected and stored in a more organised way and provides traceability of the research process, in that all the codes that are generated and aggregated can be easily traced back to the original data that was coded. See Appendix 9.5 for an example of coding in NVivo.

NVivo provides powerful search tools that allows the researcher to interrogate the data in multiple ways and levels (Welsh 2002). One such tool is the Matrix Query, which was an analysis tool used in this study. Matrix queries enable the researcher to see how often codes appear together or near other codes in the text, which can shine light on relationships between concepts that might otherwise not be visible. Matrix queries were used to see the relationships between Perceived Personal Relevance, Learning Trait and Skill Level, that led to the development of the Programming Learner Profiles which evolved to be one of the key findings in this study. See Appendix 9.6 for an example of a Matrix query.

3.6.2 Theoretical Sampling and Saturation

Theoretical sampling involves gathering data based on the concepts from the evolving theory. Decisions on which cases to interview are purposeful, depending on the stage of the research. During open coding, cases are chosen to generate the widest range of data. As the theory takes shape, cases are chosen to find dimensional range or variation of the concepts in the theory. At later stages, cases are chosen to compare and contrast with the theory (Marshall 1996; Strauss & Corbin 1998).

As much as possible, this study sought to interview as wide a range of students as possible. Due to conditions in the ethics approval, however, it was not possible to target particular students to be interviewed. Only students that responded to requests for interviews could be included in the study. Theoretical sampling was achieved by distributing flyers and requests to multiple classes, at multiple year levels, so students at all stages of the degree could be recruited.

When building Grounded Theory, the analysis continues until saturation is reached: no new concepts emerge with further interviews, categories and relationships are well developed and the theory accounts for variations (Strauss & Corbin 1998). The analysis in this study reached saturation after approximately 20 interviews. The remaining 11 interviews confirmed the theory and added some variation but did not substantially change it.
3.6.3 The Research Journey

This study evolved as it progressed. The initial goal of the study was to investigate the emotions of students that were involved in learning programming. This stemmed from the observation that while some students loved programming, other students disliked it. The researcher believed that this was because the students not only lacked the skills but were afraid of learning programming because they found it difficult.

Initially the researcher believed that if only students learnt and understood programming they would develop a liking towards it. Once the analysis evolved, it became evident after the first batch of interviews that some students would never like programming because they did not find it relevant to them. This insight led to a change in the focus of the research and to the development of the concept of Perceived Personal Relevance. It also led to the researcher challenging her initial beliefs and accepting that there will be some students that will not like programming, despite the best efforts of the teacher.

The research then expanded to ask the students questions about how they seek information to solve problems, how they work with others and also how they gauge their success. Further analysis, using matrix queries, led to the realisation that there were multiple types of students and certain traits stood out, such as perceived relevance, dependency on others and skill level. Looking at combinations of these traits led to the development of the Programming Learner Profiles. Further analysis of the differences between these profiles then led to the development of Patterns of Collaboration and Patterns of Information Use.

This research journey is an example of the researcher allowing the data to speak for the participants. It is also an example of the researcher being continuously reflective, challenging her initial beliefs and allowing the research question to evolve as the research progressed.

3.6.4 Avoiding researcher bias

The primary researcher in this study is an academic with over 15 years of experience teaching programming. This experience had to be acknowledged from the outset, as it was both a benefit and also a potential hindrance to the research. The benefit was that the researcher already understood some of the programming learning process, albeit from a teacher’s perspective, and it provided the motivation for conducting the study in the first place.
The researcher was very aware from the beginning that she had pre-conceived notions about student learning, as do all teachers, and that these notions had to be put aside and not allowed to influence the research. During the interviews, she had to allow students to describe their experiences and resist the temptation to jump in and start offering solutions.

During the analysis, the researcher had to allow the data to speak for the participants without being interpreted through a teacher’s perspective. At times, the researcher consulted her supervisors to question whether her interpretations of student’s statements were not being influenced by her biases.

The researcher believed that all students would like programming once they understood how to do it and particularly, how to problem solve and debug. The analysis that evolved from this study showed this was not necessarily the case for some students. The theory that has been developed by this study has shown the researcher that the influences on students are many and varied, and has considerably changed her views and assumptions.

### 3.7 Reliability and Validity

In order to ensure rigour and validity in this qualitative study, strategies were built into all stages of the research process: design, data collection and analysis (Morse et al. 2002; Whittemore, Chase & Mandle 2001). The strategies were: investigator responsiveness, methodological coherence, theoretical sampling and sampling adequacy, an active analytic stance, and saturation (Morse et al. 2002).

Investigator responsiveness means the researcher must remain open, use sensitivity, creativity, insight and be flexible (Morse et al. 2002). The researcher in this study ensured she was always open to concepts arising from the data and willing to challenge any pre-conceived notions. An example of this was the concept of Perceived Personal Relevance that arose during the analysis as a key influence, one which the researcher had not considered important beforehand.

Methodological coherence means ensuring consistency between the research question and the method. There may be a need to modify the research question or the research method as the research progresses (Morse et al. 2002). Grounded theory was the methodology used throughout this study, but the research question did change over time as the theory developed. The initial goal of the research was to investigate feelings and emotions towards programming, but the analysis revealed a wider range of important influences so the research question evolved to incorporate these.
Theoretical sampling means that participants need to represent the research topic being investigated. Sampling adequacy means that sufficient data is gathered to account for all aspects of the phenomenon, and saturation is reached (Morse et al. 2002). This application of this strategy is described in section 3.6.2.

An active analytical stance means that concepts arising from the analysis are compared against the data, making modification as necessary and then validating the theory against new incoming data (Morse et al. 2002; Strauss & Corbin 1998). The research plan followed in this study involved constant comparison, described in detail in section 3.3. Constant comparison is also a key part of Grounded Theory methodology (see section 3.4.1).

Saturation means that theory development should move from a micro analysis of the data to a macro theoretical understanding. The developed theory should be comprehensive, logical and consistent (Morse et al. 2002). The theory developed by this study evolved from detailed analysis to a theoretical understanding. Initial coding produced several hundred individual concepts. These were aggregated into categories and major themes. Use of matrix queries recognised relationships between three key elements in Learner Nature: Perceived Personal Relevance, Dependency and Skill Level. These were then used to define the Programming Learner Profiles, which led to a much richer and cohesive theory.

### 3.8 Chapter Summary

In order to develop a cohesive theory that gives deeper insights into the influences on the programming learning experience, this study followed a qualitative research approach and a rigorous research plan. Thirty-one students were interviewed and the transcripts analysed using a Grounded Theory methodology. The analysis developed a cohesive theory that answers the research questions and is presented over the following three chapters: 4, 5 and 6.
Chapter 4  Theory of Influences on the Student Learning Experience of Programming

As described in Chapter 3, this study explored the influences on the programming learning experience, in depth, from the student perspective. A qualitative study was conducted by interviewing 31 students and analysing the transcripts using a Grounded Theory methodology. This led to the development of a Theory of the Influences on the Student Learning Experience of Programming.

For ease of readability, this theory is presented over the next three chapters (4, 5 and 6). This chapter introduces and overviews the main model of the theory, the Influences Model of Learning Programming. The rest of this chapter discusses the outer elements of the model: Student Characteristics, Personal Networks and Information Sources. The findings related to those concepts are presented, along with student quotes where appropriate to illustrate the concepts.

Chapter 5 then presents the Student Learning Behaviours component of the model, which includes: Core Learning Perspectives and its various components, Patterns of Collaboration (including the Model of Dependency) and Patterns of Information Use.

Chapter 6 explores the core of the model, the Programming Learner Profiles. It details the concepts within Learner Nature, which define the profiles and then presents and describes seven distinct profiles that arose from the analysis: Reluctant Beginner, Willing Beginner, Keen Beginner, Budding Manager, Budding Practitioner, Budding Developer and Advanced Developer. It also explores various pathways student may take moving from one profile to another. Chapter 6 also explores the various relationships that were found to exist between the components of the model.
4.1 Model of Influences on Learning Programming

The model that was developed as a result of the analysis in this research is presented in Figure 4-1, named the Model of Influences on Learning Programming. The model is made up of the major concepts and themes that arose from the analysis and their interrelationships. Many influences on the student learning experience were found during the analysis. This model was developed to bring all those influences together in a cohesive and unified way to better understand them.
The components of the model are the Student Characteristics, Personal Networks, Information Sources and the Programming Learner Profiles. Student Characteristics describe what the student brings into the learning experience but are not part of the experience itself. Personal Networks and Information Sources are resources which are external to the students, but which all students use and interact with.

The Programming Learner Profiles capture the learning experience of programming students. The core elements of the Programming Learner Profiles are the Learner Nature, the Learning Behaviours and the interrelationships between these two. How students interact with their Personal Networks as part of the learning experience is described by Patterns of Collaboration. How students use external Information Sources during the learning experience is described by Patterns of Information Use.

Each element of the model is briefly introduced here and then expanded on over the following three chapters.

4.1.1 Student Characteristics

The characteristics of the individual students naturally have an influence on their learning as it is the students doing the learning. Student Characteristics were broken into two parts: General and Programming Context. Student general characteristics are those related to the student in general and include age, gender, stage (year) of degree, degree enrolled and working commitments outside of study. These are detailed in section 4.2.

The programming context characteristics describe what the students bring to the learning situation in terms of their preconceived expectations about learning programming, previous programming studies, prior knowledge and skills in IT, mindset and self-efficacy for programming. These are detailed in section 4.3.

4.1.2 Personal Networks

Personal networks describe the variety of people students consult or seek information from in the process of learning programming. Personal networks include a wide range of personal contacts beyond just the peers within the same class or year level. Personal networks beyond the class community include friends in the same degree but at later stage, friends outside the degree studying in the same or a different university and friends in industry or family. Personal networks also include teaching staff, including lecturers, class tutors and help lab tutors. Help lab tutors are usually students from the same degree. These are detailed in section 4.4.
4.1.3 Information Sources

The Information Sources describe the sources that students use or refer to as part of their learning programming. These include the written materials provided by teaching staff (lecture, tutorial and any additional notes). It also includes the online sources of information students refer to, such as the Google search engine, online manuals and tutorials, YouTube, professional programming forums and internal course forums. These are detailed in section 4.5.

4.1.4 Learning Behaviours

Learning Behaviours describe the intentions and behaviours of students during their learning. Core Learning Perspectives include the student’s Ownership of Learning, Learning Task Intent and Problem Solving Behaviours. Patterns of Collaboration describe the various levels of dependency students have with their Personal Networks. Patterns of Information Use describe the various ways students use the Information Sources available to them.

Core Learning Perspective

Core Learning Perspectives describes the key attitudes, intentions and behaviours of students as they proceed with their learning of programming. Core Learning Perspectives brings together three concepts: Ownership of Learning, Learning Task Intent and Problem Solving Behaviours. These concepts were found to be key influences on how students collaborate with their peers and personal networks and use their information sources.

Ownership of Learning describes the students’ views on who is responsible for their learning and understanding - whether they take responsibility for their learning or assume others are responsible for teaching them. Learning Task Intent describes the students’ intentions towards their learning tasks and deliverables - whether their priority is on understanding or just getting the deliverable done to achieve the marks. Problem solving behaviours describe the various behaviours students will carry out when solving programming problems. These range from being ad-hoc and poorly planned to thorough and thoughtful coding, error detecting and debugging.

All the concepts within Core Learning Perspectives are detailed in section 5.1.
Patterns of Collaboration

Patterns of Collaboration describe the different ways students interact, exchange information and seek assistance from others. It highlights and describes the different levels of dependency, and ultimately, how much students rely on their personal networks. These Patterns are illustrated in the Model of Dependency (see Figure 5-1) and are detailed in section 5.2.

The different levels of dependency range from a One Way Dependency where the student is highly reliant on their personal networks. In a Two Way Co-dependency, both parties are reliant or co-dependent on each other for different skills or division of labour. The Collaborative Independent and Solitary Independent interactions are more evolved. In the Collaborative Independent interaction, both parties proactively attempt their tasks and then come together to compare approaches learning of each other. In the Solitary Independent, the students help their peers but then don’t actually tend to seek help from their personal networks. Students can also interact with multiple peers or personal networks concurrently, at different levels.

Patterns of Information Use

Patterns of Information Use describes the different ways students use written and online sources as they learn how to solve programming problems. At a basic level, students look up their information sources for reference checking. This typically includes syntax verification and finding the meaning of errors. Solution discovery is looking through the sources information in search of a solution to the particular problem the student is working on, particularly assignments. The type of solution the student looks for varies on their skill level and goal. Looking for Sample Answers is the simpler form of solution discovery that involves looking for a section of code that matches the students’ problem.

Code Adaptation, Comparative Analysis and Beyond the Scope are the more elaborate forms of solution discovery. Code Adaptation involves identifying and understanding parts of subsections of code written by others that could be adapted to one’s own code. Comparative Analysis involves students comparing their code to others (personal networks and online forums) with the intention to improve their own coding style. Beyond the Scope involves using online sources to learn new languages or advanced features that are not covered in the curriculum.

Patterns of Information Use are detailed in section 5.3
4.1.5 Learner Nature

During the analysis, three distinct dimensions surfaced as being key influences in the way students approach their learning of programming, the information sources they use and their learning behaviours. These three key Learner Nature dimensions are:

- Perceived Personal Relevance (High / Low)
- Learning Trait (Independent / Dependent)
- Skill Level in Programming (Novice / Competent / Advanced)

Perceived Personal Relevance

The Perceived Personal Relevance concept surfaced during the analysis as having a major influence on how much students are interested in learning programming. For the majority of the cases, this concept describes the level of relevance the students see in learning programming in relation to their future career goals, encompassing the general interest and overall motivation to learn programming, based on how relevant they see it to their future. A minority of students mentioned having an interest in learning programming as a personal interest or as a hobby despite not intending to do it for their career.

Learning Trait

Learning Trait describes how much students rely on others for solving programming problems and their learning. Students usually seek help with planning, coding, or debugging problems from their sources of help which includes teaching staff, peers and their personal networks. Learning trait has two values: dependent or independent. These values were derived by summarising into two groups the four values from the Model of Dependency (see section 5.2).

Dependent students rely heavily on others, taking a passive approach when completing a task and iterating between their preferred sources of help. Independent students, on the other hand, minimally rely on others, showing pro-activeness with the help received. They may still use all of the information sources (written, online, peers and teachers) but they only seek a pointer or direction to help them move forward, only after having proactively attempted to solve the problem themselves.

Skill Level

Skill Level describes the programming skill level the students displayed at the time of their interview, ranging from novice, to competent and advanced.
A novice programming learner describes someone who has just begun to learn programming. Their understanding of basic programming constructs (sequence, decisions and repetition) is rudimentary. They are comfortable using simple decision and repetition constructs but combining these to build an algorithm is difficult.

A competent learner in programming is someone who has mastered all the programming constructs. They are also comfortable using simple and complex data structures. They can confidently combine the programming constructs to build a solution algorithm for a problem. An advanced learner in programming goes beyond competent level in that they can do the same but faster and more efficiently.

A more detailed description of Learner Nature is given in section 6.1.

### 4.1.6 Programming Learner Profiles

Programming Learner Profiles describe the influences on the way students go about learning programming and the way they use their Information Sources and Personal Networks. They arose from the data during the analysis and help describe the influences on the student learning experience and their patterns of behaviour in a unified and cohesive way (further elaborated in section 6.1).

The Programming Learner Profiles encapsulate the relationships and influences between the three Learner Nature dimensions (Perceived Personal Relevance, Learning Trait and Skill Level), which were used to define the profiles, and the patterns of learning behaviours (Core Learning Perspective, Patterns of Collaboration and Patterns of Information Use).

The analysis identified seven distinct profiles that describe seven different types of students and their patterns of behaviour. Each profile is an encapsulation of a number of influences interacting with each other in a unique way. Each profile has also been given a name, which helps to further understand these relationships. The profiles that emerged from the analysis are: The Reluctant Beginner, The Willing Beginner, The Keen Beginner, The Budding Manager, The Budding Practitioner, The Budding Developer and The Advanced Developer. Table 4-1 provides a summary of the seven Programming Learner Profiles. The seven profiles are described, in depth, in section 6.2.
### Development of the Programming Learner Profiles

The Programming Learner Profiles emerged (came about) by analysing individual student learning approaches to tackling programming problems, as described by each student. These were then synthesised into the seven unique Programming Learner Profiles presented earlier in this section.

In the interviews, programming students were asked to describe how they went about solving programming problems, what their experience was like and how they felt about it. During the analysis, a general tendency in the way students used their sources of information in their learning of programming became apparent. A model describing how programming students interact with their peers and social networks was first drawn from the data. In this model, it became clear that there are levels of dependency in the way students interact with their peers and social networks. Also, as the level of independence and true collaboration rises, the reliance on others goes down (this Dependency Model is further described in section 5.2).

<table>
<thead>
<tr>
<th>Profile</th>
<th>Skill Level</th>
<th>Learning Trait</th>
<th>Perceived Personal Relevance</th>
<th>Participant Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reluctant Beginner</td>
<td>Novice</td>
<td>Dependent</td>
<td>Low</td>
<td>5</td>
</tr>
<tr>
<td>Willing Beginner</td>
<td>Novice</td>
<td>Independent</td>
<td>Low</td>
<td>6</td>
</tr>
<tr>
<td>Keen Beginner</td>
<td>Novice</td>
<td>Independent</td>
<td>High</td>
<td>5</td>
</tr>
<tr>
<td>Budding Managers</td>
<td>Competent</td>
<td>Dependent</td>
<td>Low</td>
<td>4</td>
</tr>
<tr>
<td>Budding Practitioner</td>
<td>Competent</td>
<td>Independent</td>
<td>Medium</td>
<td>3</td>
</tr>
<tr>
<td>Budding Developers</td>
<td>Competent</td>
<td>Independent</td>
<td>High</td>
<td>4</td>
</tr>
<tr>
<td>Advanced Developers</td>
<td>Advanced</td>
<td>Independent</td>
<td>High</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 4-1 - Programming Learner Profiles Summary
In an attempt to compare and contrast these levels of dependency with other sources of information, a matrix query was then drawn up to pull out all the quotes relating to the use of online information sources (which at that point in time were: looking for something specific; looking online for sample code or what others do; looking for an answer or bug solution; seeking pointer or direction). To abstract themes from the quotes produced by the matrix query, a three column table was drawn up with the level of dependence and usefulness of online sources. The participants were then grouped according to: Independent – Online very useful, Intermediately Independent – Online somewhat useful, Dependent – Online Marginally useful.

In an attempt to find similarities and differences in the way each of these three main groups used online information sources, it became apparent that those participants that were dependent were novices and those who tended to be independent were more skilled. By looking at the data in this particular fashion, it became apparent that dependency had to be looked in conjunction with skill level and also motivation (which later developed into Perceived Personal Relevance). Thus these three key dimensions became the defining dimensions of the profiles used to group the participants.

A preliminary grouping of the interviews was made and initially three profiles were identified: Novice Dependent Extrinsic, Novice Independent Intrinsic and Skilled Independent Intrinsic (according to the concepts at that time). In order to pinpoint further differences between the combinations of the three key dimensions, a three dimensional table was then drawn up where further concepts from the analysis were used in order to compare similarities and differences across the three dimensions.

A preliminary grouping of the interviews was then made and every combination of the three key dimensions was mapped out, producing ten different potential profiles. All the interviews were categorised within each of these ten profiles but there were some profiles that had no interviews. The profiles were then condensed to seven and each given a descriptive name which highlights the key characteristics for each profile.
4.2 Student Characteristics - General

The characteristics of the individual students naturally have an influence on their learning as it is the students doing the learning. The next two sections explore student characteristics that were found to be influential in the student learning experience. Student Characteristics were broken into two parts: General characteristics and Programming Context Characteristics. This section covers the characteristics that are related to the student in general and not specific to the programming context.

These were: age, gender, stage (year) of degree, degree enrolled and working commitments outside of study. All the characteristics are broken down by student Programming Learner Profile to better understand their influence on the learning experience.

4.2.1 Age

As described in section 3.5.1, the students interviewed for this study were predominately younger students, which is to be expected of undergraduate students. Sixteen of the students were between 18 and 21, and a further 11 were between 22 and 29. Only 4 students were 30 years of age or older.

Table 4-2 shows a break-down of the student age groups by learner profile. It can be seen that in general, each age group is represented in each of the profiles, so no clear patterns can be observed, except that the youngest students are mostly beginners, but that is expected because they have only just started their learning.

Maturity in age, however, does seem to sharpen their Perceived Personal Relevance, making the mature age students (30 years or older) more likely to be at one extreme or the other. Those who know programming is not relevant to their career goals are more likely to be beginners and to stay that way.
“I am 30 years old. I really care about walking out of class with the sense that I’ve learnt something. I am at that stage where I’ve got to get a degree. I put a lot of commitment on studying programming, but this subject is taking too much time out of me and too much energy out of me ... it is just too hard.” (I4)

The one older student that was classified as an Advanced Developer had previously learnt programming at a younger age, and although was not working as a programmer at the time, was working in the IT industry while studying.

<table>
<thead>
<tr>
<th>Profile</th>
<th>18 to 21</th>
<th>22 to 29</th>
<th>30 to 39</th>
<th>40+</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reluctant Beginner</td>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>Willing Beginner</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>Keen Beginners</td>
<td>4</td>
<td>1</td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>Budding Managers</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>Budding Practitioners</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Budding Developers</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>Advanced Developers</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>Total By Age Group</td>
<td>16</td>
<td>11</td>
<td>3</td>
<td>1</td>
<td>31</td>
</tr>
</tbody>
</table>

Table 4-2 Participants by Age and Profile

4.2.2 Gender

As described in section 3.5.1, the study interviewed 22 males and 9 females. Although there were somewhat less females than males, this is representative of the gender ratio of students in the Information Systems degree that was the major source of participants. Also, as previously mentioned in section 3.5.1, this ratio is also representative of the gender balance in the IT industry.
Table 4-3 shows a break-down of student gender by learner profile. It can be seen that, in general, the female students are predominately Beginners and Budding Managers. As the majority of the females interviewed were first year students, it is not surprising that they mostly are in the Beginners profiles.

<table>
<thead>
<tr>
<th>Profile</th>
<th>Male</th>
<th>Female</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reluctant Beginner</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Willing Beginner</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>Keen Beginners</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Budding Managers</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Budding Practitioners</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Budding Developers</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Advanced Developers</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>22</td>
<td>9</td>
</tr>
</tbody>
</table>

Table 4-3 Participants by Gender and Profile

The females also make up 4 of the 5 Reluctant Beginners and only 1 Advanced Developer. The female that was an Advanced Developer had previously learnt programming in high school, whereas all the other females had no previous programming knowledge and only very basic previous IT knowledge in relation to basic application use (see sections 4.3.2 and 4.3.3). Most of the females in the study also reported having a low self-efficacy for programming (see section 4.3.5).

No definitive conclusion can be drawn as to the influence of gender on the learning experience, however, as the population of the study is small and may have been different with a different group of students. While females made up the majority of the Reluctant Beginners, and males made up the majority of Keen Beginners, Budding Developers and Advanced Developers, most of the females had little prior programming experience whereas the males in the more advanced profiles tended to have prior knowledge of programming.
4.2.3 Degree Enrolled

Table 4-4 depicts the degree the students were enrolled in, or graduated from, at the time of the interview, by profile. Almost two thirds of the students in this study were enrolled in the degree of Business Information Systems (20 of 31) and one fifth of the students were graduates from the same degree (6 of 31).

About half of the students enrolled in Business Information Systems were Beginners, most of these being Reluctant and Willing Beginners. By contrasting this table with the Preconceived Expectations table (see section 4.3.1), most of these Reluctant and Willing Beginners have the preconceived expectation to learn minimal programming. Two of the six graduates from the Business Information Systems degree remained at the Beginner level (one Reluctant and one Willing), even after graduating. Another two graduates plateaued as Budding Managers.

Interestingly, the majority of the students from non-IS degrees (Accounting, Marketing and International Business) were Keen Beginners and Advanced Developers. This may be because these students chose to learn programming as an elective because they have a higher Perceived Personal Relevance toward programming. They described an interest in learning programming as they saw it useful and relevant to their careers, even though they were not planning to be full time programmers.

<table>
<thead>
<tr>
<th>Profile</th>
<th>Info Sys</th>
<th>Info Sys Grads</th>
<th>Accounting</th>
<th>Computer Science</th>
<th>Marketing</th>
<th>International Business</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reluctant Beginner</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Willing Beginner</td>
<td>5</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Keen Beginners</td>
<td>3</td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Budding Managers</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Budding Practitioners</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Budding Developers</td>
<td>3</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Advanced Developers</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>20</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 4-4 Participants by Degree Enrolled and Profile
4.2.4 Stage in Degree

Table 4-5 shows a break-down of the stage (year) of their degree the students were in at the time of their interview, by learner profile. Half of the participants in this study were doing their first or second year of their degree, while just over a quarter were in their final year. Just less than a quarter were graduates.

<table>
<thead>
<tr>
<th>Profile</th>
<th>Stage (year) in Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>First</td>
</tr>
<tr>
<td>Reluctant Beginner</td>
<td>3</td>
</tr>
<tr>
<td>Willing Beginner</td>
<td>1</td>
</tr>
<tr>
<td>Keen Beginners</td>
<td>3</td>
</tr>
<tr>
<td>Budding Managers</td>
<td>1</td>
</tr>
<tr>
<td>Budding Practitioners</td>
<td>1</td>
</tr>
<tr>
<td>Budding Developers</td>
<td>2</td>
</tr>
<tr>
<td>Advanced Developers</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 4-5 Participants by Stage in Degree and Profile

Most of the Beginner profiles were first and second year students, which is to be expected, as the participants in this study mostly do programming courses in those years of their degree. More interestingly, some final year students were still at the Beginner stage. One of these was enrolled in a non-IS degree and were doing programming as an elective in their final year. There was one final year student, however, who remained a Reluctant Beginner, despite having passed several programming courses, as well as two graduates who plateaued at the Beginner Skill Level even years after graduating.

On the other hand, there were 3 first year students who were Keen Beginners and a number of second year students already at the Budding and Advanced Developer stage. This shows that students with a high Perceived Personal Relevance are able to move quickly up the profiles. These types of students also tend to possess good problem solving skills to start with, usually because they have already learnt programming on their own or in high school.
One third of the students were from the Budding profiles, mostly from second and third year as well as graduates. This is also to be expected as some students will tend to progress in second and third year as they learn more, refine their skills further and form personal networks within their class/year level where they learn from each other.

### 4.2.5 Working Commitments Outside of Studies

Table 4-6 depicts the number of hours full time and part time students described being employed to work, outside of their studies. Some of these work commitments include personal projects. One third of the students did not report the number of hours they do employed work.

The most interesting trend to notice was that half of the Budding and Advanced Developers, who were also full time students, reported working between 20 and 29 hours per week. Three of these students (One Budding Developer and Two Advanced Developers) reported working on their own programming projects on the side while doing their studies. This is a qualitative characteristic of these two profiles as these students seek out to further extend their skills by doing additional projects.

<table>
<thead>
<tr>
<th>Profile</th>
<th>Full Time Students Working Hours</th>
<th>Part Time Students</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Not reported</td>
<td>1-19</td>
</tr>
<tr>
<td>Reluctant Beginner</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Willing Beginner</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Keen Beginners</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Budding Managers</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Budding Practitioners</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Budding Developers</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Advanced Developers</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>13</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 4-6 Participants by Study Load, Working Hours Outside Studies and Profile
4.2.6 Summary

The characteristics of the individual students would naturally have an influence on their learning as it is the students doing the learning. This section covered those related to the student themselves: age, gender, stage (year) of degree, degree enrolled and working commitments outside of study. Each concept was broken down by Programming Learner Profile.

In this study, each age group was represented in each of the profiles, so no clear patterns can be observed. One trend to note was that maturity in age may sharpen the student’s Perceived Personal Relevance, making the mature age students more likely to be at one extreme or the other.

Although there were somewhat less females than males in the study (22 males and 9 females), this was representative of the gender ratio of students in the Information Systems degree that was the primary source of participants. Females in the study made up 4 of the 5 Reluctant Beginners and only 1 Advanced Developer, but no definitive conclusion can be drawn from this as the population was small.

The vast majority of the students in this study were enrolled in or had graduated from the degree of Business Information Systems. Interestingly, the majority of the students from non-IS degrees (Accounting, Marketing and Business) were Keen Beginners and Advanced Developers. This may be because these students chose to learn programming as an elective and saw it as relevant to their future.

While half the students worked while studying, the most interesting trend to note was that half of the Budding and Advanced Developers, who were also full time students, reported also working between 20 and 29 hours per week.
4.3 Student Characteristics – Programming Context

This section describes the characteristics the students bring to the learning situation that are specific to the programming context, including their preconceived expectations about learning programming prior to commencing their studies and the prior knowledge and skills they had acquired in relation to programming and IT. The students’ mindset as well as their self-efficacy beliefs for programming and IT, are also discussed.

All the characteristics described in this section are broken down by student Programming Learner Profile to better understand their influence on the learning experience.

4.3.1 Preconceived Expectations

Preconceived expectations describe the initial expectations students had about learning programming prior to commencing their degree. These were summarised into: no expectations, expected to learn something useful, expected it to be difficult and expected to learn programming in less depth. Table 4-7 depicts the preconceptions broken down by student profile. This section discusses these expectations and contrasts them to the actual experiences the students had in learning programming.

One fifth of the students had no expectations about learning programming (6 out of 31). Most of these students had a positive experience with learning programming. Two of these students however, who were Reluctant Beginners, found the experience of learning programming difficult because they found programming foreign. They also found it difficult learning to work out coding problems by themselves instead of being given an answer to refer to, which in part were related to difficulties experienced adapting to a university environment, being quite different to high school.
“To be honest I had no idea, because my first year was all common core business subjects. I didn’t really enjoy it much and I thought I might have been in the wrong degree but it was only that one programming subject I was exposed to until I started realising, ‘Oh okay. Now, now I get the rope.’” (I27)

<table>
<thead>
<tr>
<th>Profile</th>
<th>Preconceived Expectations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>None</td>
</tr>
<tr>
<td>Reluctant Beginner</td>
<td>2</td>
</tr>
<tr>
<td>Willing Beginner</td>
<td>2</td>
</tr>
<tr>
<td>Keen Beginners</td>
<td>3</td>
</tr>
<tr>
<td>Budding Managers</td>
<td>1</td>
</tr>
<tr>
<td>Budding Practitioners</td>
<td>1</td>
</tr>
<tr>
<td>Budding Developers</td>
<td>3</td>
</tr>
<tr>
<td>Advanced Developers</td>
<td>4</td>
</tr>
<tr>
<td>Total</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 4-7 Preconceived Expectations of Participants by Profiles

Almost half of the students described expecting to learn something useful in relation to their future career goals. For the majority of these students, their experience met their expectations. These students also have a high Perceived Personal Relevance to learn programming. Of the 13 students who expected to learn something useful, 4 (of 31) were Beginners, mostly Keen Beginners, 5 (of 31) were from the Budding profiles mostly Developers and all of the Advanced Developers 4 (of 31). One of the Advanced Developers expected to learn more about IT, including programming.

“I was expecting to learn a bit of IT (administration and networking) and a bit of coding, which is what I’ve started to do. I’m also expecting to work with people in a more business sense advising people. That’s really where I would go with this after I’ve finished the degree.” (I13)
“It is a very powerful thing to know, considering everyone uses the end product of programming. I found learning programming, the ability to develop solutions using programming very relevant and it’s not just for my career goals, but for things outside of my own career like family business and stuff like that.” (I22)

One sixth of the students expected to find learning programming difficult (5 out of 31). Interestingly, most of these students found the experience of learning programming doable and were happy with what they had learnt.

“Prior to learning programming, I assumed it would be complex because I did not know how it actually worked or how code is written. But once I started the degree and got some experience, I understood the basics and the complexity started to shrink.” (I2)

Only one Reluctant Beginner found the experience of learning programming difficult as expected, however, the negative expectation was influenced by a negative experience prior starting the degree and also this student had a low Perceived Personal Relevance even from high school.

“My first experience in secondary school was very scary and unexpected. I never knew what programming was about. I did not understand it and I thought that was not the kind of area I would go for at that stage. I am still not confident enough with the practical work, but I can talk about the theory”. (I)

Less than one quarter of the students expected to learn programming but in less depth (7 out of 31). These students are almost evenly spread between the Reluctant Beginners, Willing Beginners, Keen Beginners and Budding Managers. The majority of these students found the experience of learning programming hard, in particular coding and debugging. The Reluctant Beginners also found it hard learning to work through coding problems themselves.

“SQL as a language just made sense to me as opposed to some of the other programming subjects I just couldn't grasp quite as well. Concept-wise, I understood everything, but I found the implementation and debugging very heavy.” (I)

“I thought there would be a little bit of programming. I didn't realise that how deep it was. When I first started, I didn't realise we were going into CRAZY programming. It was hard being able to write and understand the code you’re writing.” (I21)
Lastly, there was one Budding Developer who started off with the overall expectation to learn something useful but expected to dislike programming based on previous experience. However, for this unique case in this study, the student came to understand the relevance of what he was learning and his Perceived Personal Relevance changed from low to high during his first programming course.

“I’ll be honest, in high school I was more interested in web design. Originally, I had no interest in programming. I thought I’m never going to do this again. When I discovered the intro course had programming in it, I was thinking, ‘I’m going to hate this’… After doing the programming subjects and going to design my own stuff, I found I got a real interest because I thought, ‘Well if I can do it in here, I’m sure if I apply myself, I could do it outside.’” (118)

### 4.3.2 Previous Studies of Programming

The section describes what studies of programming the participants had previously completed in high school or diplomas (see Table 4-8).

<table>
<thead>
<tr>
<th>Profile</th>
<th>No Previous Studies</th>
<th>Previous Studies in High school or Diploma</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reluctant Beginner</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Willing Beginner</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>Keen Beginners</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Budding Managers</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Budding Practitioners</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Budding Developers</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Advanced Developers</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>Total</td>
<td>20</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 4-8 Participants Previous Studies of Programming by Profile
Only one third of the students in this study learnt programming prior to commencing their degree (11 of 31). Most of these students are from the Advanced Developers, Budding Developers and Budding Practitioners profiles. These students described their previous programming experience as a positive one and displayed a higher interest and Perceived Personal Relevance in programming, even prior to commencing their degree.

Two of the students who learnt programming previously described a previous negative experience (one Reluctant Beginner and one Willing Beginner). The Reluctant Beginner’s previous negative experience was the most profound, as the student described being frightened of learning programming. This student also described low self-efficacy towards programming and a low Perceived Personal Relevance and remained a Reluctant Beginner, even years after graduating.

In one case, a Budding Developer who had an earlier negative experience and disliked programming, developed an interest in programming when he was able to see how he could bring together an understanding of programming to solve real business problems. This student’s Perceived Personal Relevance switched from low to high during their first programming course.

“I found that I picked it up a lot better because there was so much more help and support materials, compared to high school. I actually went off and started designing some software of my own. I could see some sort of purpose in a business. I didn’t really see that earlier.” (I18)

Two thirds of the students did not learn any programming in earlier studies (20 out of 31). The majority of these students are from the Reluctant Beginners, Willing Beginners and Budding Managers profiles. Most of these students, however, did report previously studying maths and/or IT in high school or diploma. Some of these students mentioned that even though they had done maths in high school or diploma, and were comfortable with maths, they still found programming difficult.

The quality of a student’s previous studies with programming does seem to influence their Perceived Personal Relevance and learning experience. Those with a positive previous experience are more likely to have a higher Perceived Personal Relevance and reach the more competent profiles. Those with a previous negative experience, on the other hand, are more likely to stay reluctant and not be interested in learning programming.
4.3.3 Previous IT Knowledge or Skills

Table 4-9 depicts the general IT skill level or knowledge students brought into the degree, by profile. Students were categorised into three main groups: those that had studied IT previously or had industry experience, those that had done some IT in high school and those that only had a basic understanding.

<table>
<thead>
<tr>
<th>Profile</th>
<th>Diploma or Practice experience</th>
<th>General IT from High school</th>
<th>Basic Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reluctant Beginner</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Willing Beginner</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Keen Beginners</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Budding Managers</td>
<td></td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Budding Practitioners</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Budding Developers</td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Advanced Developers</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>6</td>
<td>17</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 4-9 Participants Previous IT Knowledge / Skills by Profile

One fifth of the students (6 of 31) had an in-depth previous knowledge of IT, acquired either through doing a prior diploma in IT or through practice experience gained through working with IT or independent learning. Most of these students were already familiar with programming and also with hardware. The majority of these students were in the Beginners profiles. The one Reluctant Beginner who completed a diploma described an overall perception of difficulty with programing and a disinterest to pursue it as a career goal.

Over half of the students (17 of 31) had an intermediate level of prior IT knowledge through completing IT subjects in high school. Over half of these students were from the Budding and Advanced profiles. The majority of these students discussed learning mostly general IT and doing minimal programming, such as using HTML or Dreamweaver to create a website. Two students in the advanced profiles described learning programming independently and going beyond what was expected of them in high-school.
One quarter of the students (8 of 31) had a basic applications skill level. This consists of students having used software applications such as Microsoft Office, Photoshop or Dreamweaver to create websites but with no coding involved. All of these students were from the Beginner or Budding Manager profiles.

The general theme that is observable is that previous IT knowledge does, in most cases, influence the learning of programming. Having previous IT knowledge and skills enables the students to feel more comfortable with learning programming and progress through the profiles more quickly. All the students that were in the higher competent profiles (Budding Practitioner, Budding Developer and Advanced Developer) had studied IT in a diploma or at least done some IT in high school. Most of the Reluctant Beginners and half of the Budding Managers, on the other hand, only had basic previous IT skills.

### 4.3.4 Mindset

To gain a better understanding of the students’ mindset, students were asked how they gauge their success in programming. According to Dweck (2008), an answer describing marks as the measure of success signals a fixed mindset and an answer that describes understanding as the measure of success signals a growth mindset. Table 4-10 depicts how the students in this study reported gauging their success. Their answers have been mapped against the profiles the students fall into.

<table>
<thead>
<tr>
<th>Profiles</th>
<th>Marks Fixed Mindset</th>
<th>Understanding Growth Mindset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reluctant Beginner</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Willing Beginner</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Keen Beginners</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>Budding Managers</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Budding Practitioners</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Budding Developers</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Advanced Developers</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>Total</td>
<td>11</td>
<td>20</td>
</tr>
</tbody>
</table>

Table 4-10 - Mindset of Participants by Profile
One third of the students described their measure of success being marks (11 out of 31). Most of these students were from the Reluctant Beginners, Willing Beginners and 2 from the Budding Managers profiles. These students displayed a fixed mindset not only by how they gauged their success, but also by the way they described learning programming.

Half of the Budding Managers measure success by the marks 2 (out of 31). These Budding Managers plateau at a basic understanding and do not see mastery in programming as relevant to their future careers. Spending time working on complex programming tasks is seen as time consuming and not relevant.

Two thirds of the students described understanding as being their measure of success (20 out of 31). These include half of the Willing Beginners and all of the Keen Beginners, Budding Practitioners, Budding Developers and Advanced Developers. These profiles display a Growth Mindset towards programming, not only by how they measure their success but also by all their other descriptions as to how they deal with problems, treating them as challenges and persevering for much longer periods.

“I was really keen to just get into the thick of it and really I breezed through it fairly easily and I ended up helping others. I think that’s because I had prior experience but I also had the interest to keep learning. I can understand why people just sort of give up and get really frustrated but I’m the type of person that I love a challenge and I love to work through a problem.” (I14)

Further analysis of the answers to this question revealed that some students actually have a mixed or multiple mindsets. They described having a growth mindset in areas they have a high Perceived Personal Relevance and a fixed mindset in areas they have a low Perceived Personal Relevance. For example, some of the students that display a fixed mindset towards programming actually display a growth mindset towards other IT areas that are less programming oriented. It also coincides that these students have a low Perceived Personal Relevance for programming. They don’t see placing effort and mastering programming as relevant, yet they embrace other IT areas such as multimedia, databases and networking.

“Even after a year and a half, I was still fumbling my way through programming and it just felt like failure, failure and failure. With multimedia however, I would stumble, but the stumbling was always driving you further and pushing you further to make something better. With programming, you were just trying to get something to a standard which was just working, as opposed to trying to get something that was excellent.” (I11)
4.3.5 Self-Efficacy for Programming and IT

Self-efficacy as a person’s belief in their ability to succeed in specific situations or accomplish a task (Bandura 1997, 2002). Self-Efficacy beliefs influence the choices a person makes such as effort spent on a task, persistence and their responses to obstacles when completing a task, despite ability and skill.

In this study, self-efficacy beliefs were gauged based on the students’ descriptions with respect to programming and also with their descriptions towards other IT related areas. Comments such as “I do not feel very talented to do as well” and “I did not think I had the skills to do programming” were treated as descriptions of low self-efficacy. Descriptions such as “there was nothing where I felt I was out of my depth” were treated as high self-efficacy.

Table 4-11 depicts the self-efficacy beliefs of the participants in this study broken down by profile.

<table>
<thead>
<tr>
<th></th>
<th>High Self-Efficacy</th>
<th>Low Self-Efficacy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Programming Only</td>
<td>Programming and in IT</td>
</tr>
<tr>
<td>Reluctant Beginner</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>Willing Beginner</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Keen Beginners</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Budding Managers</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Budding Practitioners</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Budding Developers</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Advanced Developers</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>Total</td>
<td>1 Female 3 Males</td>
<td>2 Females 14 Males</td>
</tr>
<tr>
<td></td>
<td>5 Females 5 Males</td>
<td>1 Female</td>
</tr>
</tbody>
</table>

Table 4-11 Participants Self-Efficacy for Programming and IT by Profile
All of the Reluctant Beginners and two thirds of Willing Beginners described low self-efficacy beliefs with programming. These two profiles also tended to have a low Perceived Personal Relevance, so they did not see putting effort and perseverance in developing their programming skills as relevant to their future career goals.

“The lecturer explained it to me ... but I still do not get it. I’ve come up with the term ‘you either get it or you don’t’ or maybe I am just too dumb for programming. Maybe my mind is not good for programming.” (I4)

“I always thought programming is something very difficult. Personally I am not keen to be a programmer so maybe I did not work hard enough to be a programmer. I found some other areas such as USE CASE analysis very interesting. I use that now working in administration.” (I1)

The low self-efficacy however, does not extend to other IT related areas, in most cases. Five of the Reluctant Beginners with a low self-efficacy in programming described having a high self-efficacy with Databases and HTML. These students were all females. One Reluctant Beginner, three Willing Beginners and one Budding Manager with a low self-efficacy in programming described a high self-efficacy with IT in general. These students were all males.

“I really enjoyed Database. That probably was my favourite programming subject. SQL as a language just made sense to me as opposed to some of the other programming subjects, I just couldn’t grasp quite as well. It’s the fact that I wasn’t good at them – that’s why I didn’t like them.”(I16)

Some Willing Beginners and Keen Beginners described a high self-efficacy with programming and also with other IT related areas. These students also have a high Perceived Personal Relevance with programming. The Keen Beginners who described a high self-efficacy with programming, but not with IT in general, are students from Business courses doing programming as an elective. These students, despite having a high Perceived Personal Relevance for programming which is linked to their Business career goals, do not intend to become programmers as such. These students are also in the process of developing their IT skills but not necessarily to become IT practitioners.

Half of the Budding Managers, All the Budding Practitioners, Budding Developers and Advanced Developers have a high self-efficacy towards programming and IT in general, which is not surprising.
Most of the students with a high self-efficacy also displayed a growth mindset towards programming. Similarly, the students with a low self-efficacy displayed a fixed mindset. This overlap between Mindset and Self-Efficacy seems to present itself in each respective profile for both fixed and growth mindsets, but it is not as clear cut in the students who display a mixed or multiple mindsets. The students who display mixed or multiple mindsets are mostly students from the Budding Managers, half of the students from the Willing Beginners and one Reluctant Beginner. Half of the Budding Managers have a high self-efficacy for both programming and IT, whereas the rest of the Budding Managers have a low self-efficacy for programming only but a high self-efficacy for non IT related areas. This observation also occurs with the Willing Beginners.

“In general there was nothing where I felt I was out of my depth with programming. I knew there was a solution; I just couldn’t find it. I felt stressed maybe, not angry. Maybe frustrated because most of the time it was something simple that I had overlooked.” (I29)

4.3.6 Summary

Programming context student characteristics describes what the students bring to the learning situation in terms of their preconceived expectations, previous programming studies, prior knowledge and skills, mindset and self-efficacy.

In relation to preconceived expectations, 6 students had no expectations. While most of these had a positive experience, two experienced difficulties. 13 students expected to find it useful and most of these students met their expectations. 5 of the students expected programming to be difficult, but only one of these actually found it to be as difficult as they expected. 7 of the students expected to learn programming but in less depth than they actually did. The majority of these students ran into difficulties.

When it came to previous studies, one major trend noted was that most of the students that had studied programming previously were from the more competent and advanced profiles, while the majority of Reluctant Beginners and Budding Managers had no previous studies in programming. The quality of previous studies does seem to influence their Perceived Personal Relevance, with a positive previous experience leading to a higher Perceived Personal Relevance.
Previous IT knowledge does also influence the programming experience. All the students that were in the higher competent profiles (Budding Practitioner, Budding Developer and Advanced Developer) had studied IT in a diploma or at least done some IT in high school. Most of the Reluctant Beginners and half of the Budding Managers, on the other hand, only had basic previous IT skills.

Students were asked how they gauge their success in programming and grouped into fixed and growth mindsets based on their answers. Not surprisingly, the majority of the students showing a fixed mindset were from the Reluctant and Willing Beginner profiles, while the majority of students with a growth mindset were from the Keen Beginners and more competent student profiles.

The interesting observation regarding Mindset in this study was that some students actually have a mixed or multiple mindsets. Some of the students that display a fixed mindset towards programming actually display a growth mindset towards other areas such as multimedia, databases and networking.

All of the Reluctant Beginners and two thirds of Willing Beginners described having a low self-efficacy with programming, which was not surprising. Similarly, all of the more competent profiles had a high self-efficacy. Most of the students with a high self-efficacy also displayed a growth mindset towards programming. Similarly, the students with a low self-efficacy displayed a fixed mindset.

The low self-efficacy however, does not necessarily extend to other IT related areas. Five of the Reluctant Beginners (all females) with a low self-efficacy in programming described having a high self-efficacy with databases and HTML. This connects with the observation that students can have a mixed or multiple mindsets.
4.4 Personal Networks

In this study, the majority of the students described asking for assistance or consulting another student whenever they came across coding problems, both in their learning tasks as well as their assignment deliverables. Initially, most students would seek help from other students whenever they had a coding problem and the tutor was unable to help. These were irregular and impromptu interactions on a needs basis. For some of these students, their interactions with their peers became more regular, mostly taking place outside the classroom.

Some of these more regular interactions evolved into a personal network. Students also discussed seeking help from other personal contacts, friends or personal contacts from later years in the same degree or from a different degree. They also described consulting friends from industry or from other universities and even family members. As expected, students also seek help from teaching staff such as lecturers, class tutors and help lab tutors.

This section describes the range of personal networks that students participate in as part of their learning of programming. All student profiles interact with the personal networks described in this section, but in different ways and with a different order of preference. These differences between profiles are described in Patterns of Collaboration (section 5.2). The personal networks preferred by each student profile are described in Table 6-2 - Profile Preferences for Collaboration, found at the end of section 6.4.3.
4.4.1 Personal Networks within the same Class / Year Level

Over half of the students in this study described asking peers in the same class for assistance with coding problems, especially when they first started learning programming (15 out of 31). The most common reason given was to seek guidance or a solution for a coding problem if the tutor was not able to help them at that point in time. Initially, students would consult the person sitting next to them during the tutorial class, especially earlier on in the first semester of learning programming.

“I’d chat to the guy next to me. We would chat about what we were doing and look at each other’s program, but really superficially. Occasionally, we would help each other with loops. ‘We were not officially helping each other.’” (I13)

These interactions started off being informal and irregular, but these often developed into more regular patterns of interaction, as students developed stronger relationships and friendships with each other.

“In the first programming course it was maybe from the third class onwards. I was lucky because I had some people around me that were really good at programming and helped me out quite a bit.” (I16)

In some cases, the peer being consulted had a similar competency in programming as the student asking for help or seeking to discuss a problem with (11 out of 31).

“One good thing I did have a good friend in the class and we worked well together helping each other out with working out errors and things we didn't understand. I think we both had similar attributes and problems.” (I21)

In other cases, students specifically sought assistance from peers and friends from the same class who were perceived to be more knowledgeable and have a higher skill level (9 out of 31).

“I remember feeling overwhelmed with some of the assignments and having no idea how to solve it and that is when I saw someone in the lab that was in the course that was doing well.” (I10)

While some of these consultations took place in the classroom, the majority of the discussions and consultations were described to have taken place outside the class room and evolved to be of a more regular nature. Many of these regular consultations outside the classroom took place face to face or via online media (e.g. text messaging, Facebook or email).
“With my mate, we’d often go on Facebook and talk about code and he’d say, ‘Can’t get my data grid view to load’ and I’d send him through my little chunk of code. Then he’d debug it and go, ‘Have you tried doing this for your code?’ I mean that sort of collaboration. It wasn’t about giving the answers, it’s about working together and understanding.” (I18)

What may have started as a necessity of seeking assistance with a coding problem evolved to a co-dependent or even a collaborative type of interaction later on. Half of the 15 students (out of 31) who sought help from peers described progressing to regularly working co-dependently with these peers. A number of students described being part of a formal study group (11 of 31). These study groups commenced either towards the second half of their first programming course or in their second year. The study groups were described to be more formal in nature because students would organise to meet regularly after class to work not only on programming but also on other courses. They described encouraging each other to finish homework, discussing problems together, learning from each other how to do things and in some cases co-dependently working on coding problems and exchanging code with one another. Half of the study groups were described to have student partners with the same skill level in programming. The rest were described to have partnered with students who had a higher programming skill level.

“We have a study group. When we were at home, we would communicate probably just with Facebook messaging or social networking. At university, we would get together during our breaks, each one on our laptops and we would just work on our own projects. If someone hit a problem, we would discuss it, so we helped each other out.” (I26)

Six of the students in this study described being the students being asked for help and sharing their knowledge and helping out other students, both inside and outside the classroom and on the internal forum as well.
4.4.2 Wider Personal Networks

This section describes the different types of friends or personal contacts students described seeking information from who were outside their year level or degree. These interactions took place outside the classroom and were done either face to face or over email or Facebook. Students discussed seeking programming help from:

- Friends in the same degree
- Friends in a different degree
- Friends in industry or family

Friends in the Same Degree

Four of the students in the study described consulting friends or personal contacts that were in the same degree but that were at a later stage or year in the degree. Generally these were students they worked with and became friends with in other courses. In some cases, students had befriended the help lab tutors and continued to seek their assistance with problems even when they no longer worked in the help lab.

“Outside the tutorial, I had a friend that actually studied the year before and he tutored me for a couple of times if I had some really big issue that I couldn't overcome so he was sort of my ‘go to person’. I actually met him in one of the help labs. He showed me what he had been taught in his co-operative year which was quite a good way of debugging.” (I16)

Friends in a Different Degree

Overall seven students in this study described consulting with their wider personal networks outside their degree (7 out of 31).

Five of the students in this study discussed consulting friends from a different university who were doing either a similar degree to them or who were doing a postgraduate or a master degree.

“I actually emailed my friend in America, he is a real programmer and he is doing his Masters over there.” (I4)
Some students who consult their wider personal networks (outside their degree) do so because they trust their friends understanding about programming. This is particularly noticeable in the Advanced Developers profile as these students find it difficult to consult their peers who are not at par with them regarding advanced issues with programming.

“I would only probably contact my friends from high school that we were together with because I knew they were good programmers too.” (I8)

**Friends in Industry or Family**

Three students in this study discussed seeking programming help from friends who were working in industry. Only one described consulting a friend working overseas.

“I have a friend I went to school with who is in the industry. He actually is a systems architect. He has a Computer Science background and he is pretty good at IT. Even though he doesn’t know Visual Basic, he understands. When I ask ‘What do I do?’ he suggests, ‘Just do the pseudo-code and write out what you need to do’.” (I29)

Also, only one student described consulting / having discussions with a family member who had knowledge in programming.

“Mostly I heard from my brother who did coding at a young age that Java is widely used in industry, but he’s not in programming. Also, when I hear about young people who just got into the industry and they are doing a project, the majority of them use Java.” (I22)

In this study overall 26 (out of 31) students described consulting / using their peers, friends or personal contacts (personal networks) as an information source when attempting to solve a programming coding problem.
4.4.3 Teaching Staff

Teaching Staff, as a source of information, includes the Lecturer, Tutors and help Lab Tutors.

Help Labs

Help labs are additional labs outside the class time (lecture and tutorial) where students can drop in for assistance with either a learning task or deliverable. The help labs used by the students in this study were run by senior student tutors and provided help with a range of IT courses, including programming. The student tutors are students in the later years of the same degree who have achieved high results and are also proficient with programming. Half of the students in this study described using the help labs as an information source in the process of learning how to program and for assistance when faced with coding problems (15 of 31).

The help lab student tutors are peers from the same degree as the students seeking assistance but they are from a different Programming Learner Profile. The students interviewed in this study that had worked as help lab tutors were from the Advanced Developer and Budding Developer profiles, whereas the students seeking assistance were from the Beginner profiles (Reluctant, Willing or Keen), or from the competent profiles (Budding Managers or Budding Practitioners).

Half of the students attending the help labs described attending on a regular basis, more than 3 times in a semester (7 of 31). A number of these regular help lab attendants continued regularly attending well beyond their first semester (4 of 31). These regular help lab attendants described seeking assistance with getting started (planning and structuring an algorithm) and also with debugging issues, particularly with deliverables such as assignments.

“I went to the help labs throughout the whole thing. I used to go to one every week. The help lab tutor was probably the best at explaining actually how to go about it. Sometimes, I would not even really know where to start because our notes would only go so far.”

(I16)

“After a while, if I analysed the code and I couldn’t find anything wrong, I would just give up. But when I went to university, I would go to the Help lab to ask them where I’ve gone wrong. Otherwise I would go to another tute.”

(I27)
Other students attended the help labs sporadically, attending less than three times in a semester and mostly during their first year of learning programming (8 of 31). They described needing to attend these help labs when they felt they were falling behind with either the lecture or tutorial material, or when they felt they did not know how to get started with a deliverable or were just falling behind with it.

“I went to the help labs once in the semester. The help lab tutor helped me with the assignment. Sometimes when you fall behind it gets a bit daunting because things start to pile up and you can’t fall behind. Mainly dealing with the second and third assignment and the class work it got a bit hard and stressful to do both. My attitude in first year was ‘It’s pretty hard. I’m just going to relax a bit and then try later.’” (I20)

“I think I went to a couple of Help labs before the assignment too. There were special classes for the assignment as well, so I did catch-up classes. Only if I left the assignment or lab work too late close to the deadline and it was frustrating not being able to get my pieces of code to work. I also went to a couple of the Help labs to complete the tutorials (learning tasks).” (I23)

Some students will continue to consult help lab tutors outside of the help labs and even when they no longer work in the help lab – they become part of the students’ wider personal networks.

**Lecturers and Tutors**

Generally students described preferring to consult different teaching staff for different scenarios. Students described asking the tutor in class whenever they faced hands on problems that required the tutor to go through the code with them for them to understand. In the tutorial class, two thirds of the students in this study would seek help from the tutor with coding issues (26 out 31). Only six students described asking the tutor for how to get started with a deliverable.

“In class I would ask particularly the tutor because he had the time. He would sit right next to you and help you. He used to give us methods of how to do it. He drew it out on paper how to do the planning. The translation into code was a bit tricky.” (I26)
Students described following up issues with the lecturer, particularly for clarifying assignment specification, marking issues or for concepts discussed in the lecture but not for hands on coding problems unless the lecturer was also their tutor (18 out of 31). They also described consulting lecturers in formal consultation time slots with the lecturer for serious problems they could not solve independently or to seek a pointer or direction if working ahead or on an advanced feature (9 out of 31). Some students also described attending Q&A sessions run by the lecturer to help with assignment deliverables (2 out of 31).

“Sometimes the lecturer would run a backup-group session to guide us through the issues we need to address and to give us hints to get through it.” (I1)

4.4.4 Personal Network Considerations

Convenience

Students rely heavily on what is the most convenient information source depending on the type of issue they have, the urgency and the time they have to resolve the issues.

“To seek help from the lecturer you have to make an appointment or send an email so some students prefer to ask in the tute.” (I17)

If the issue the student is having is perceived to be more than a simple straightforward question, students are more inclined to attend help labs instead of asking the lecturer / tutor or making the effort to make an appointment to speak to the lecturer.

“No, I never thought to ask the lecturer, to be honest. Just because I feel like you’d have to be in front of a computer and coding it and have someone sitting with you. I think it’s more of a hands-on thing and also there’s so much I didn’t understand, I don’t think my questions would have been completely answered in one half an hour session. I needed constant help!” (I16)
Intimidated - Fear of Judgement

A few students (2 out of 31) felt intimidated asking teaching staff, particularly the lecturer but also with tutors, because of fear of judgement. One student described only being confident in approaching the lecturer together with other study group members, believing that lecturers only help with overview ideas but not with hands on issues. The other student described seeking assistance from the tutor only as a last resort after having tried using other sources of information such as notes and online forums.

“I look at my notes and online forums prior asking my tutors. They are my last point because I don’t want to waste their time if I can figure it out by myself. Especially if it is something I can just find out.” (I20)

Confident approaching teaching staff

Students who described being very confident in following up issues with the lecturers and tutors were those students in the Budding and Advanced Developer profiles. They usually sought help when they had advanced issues beyond the scope of what they were learning. In this scenario these students had already previously researched the topic prior consulting teaching staff (6 out of 31).

Novice Students prefer Face To Face

Those who are interested also learn by observing their tutor’s tactics for how to handle different issues.

Most of the novice students from the Reluctant Beginner and Willing Beginner profiles prefer a face to face explanation as they seek a richer explanation for their coding issues. They described finding it very difficult to visualise and comprehend the help that is given over on-line mediums particularly if it involves identifying what is causing the coding problem.

“If you don't know what is broken or what the issue is, interacting out of a forum can be difficult versus being face to face and saying to the lecturer/tutor this is what I have done, this is what I have tried and him being able to take what you are asking and come back to you.”(I9)
As these students are learning programming for the first time, they lack the mental model to understand the guidelines and steps given to them over other mediums that lack the interactive response and guidelines/explanation from the teacher.

“I preferred to ask the tutor or lecturer because they specialise in teaching. They understand how to get the concept to a student who has no idea. A student would say ‘Well this is how I got the answer’ and not explain it because they don’t know how to explain it to somebody who’s got no idea.” (I25)

Another motivation which prompts students to seek an interactive consultation with the tutor is to learn by observation how an experienced programmer goes about using the debugger diagnostic tool. In this case the student wants to go beyond just solving the coding issue and learn an effective technique that will help them solve future problems independently.

“When I had a coding problem and did not understand why it was not working I saw how the Lecturer / Tutor stepped through the code step-by-step saying, ‘This is what’s happening. This is why it’s not working. Try this, see if that works.’” (I18)

“When I got shown how to use the debugger I was like, ‘Where’s this been? I could have done with it at other points,’ to know that the variables were actually getting used properly.” (I29)

Consulting Teaching Staff as a Last option

Students in the Budding and Advanced Developers profile described using teaching staff as a last resort (9 out of 31). Four of these students described initially relying on the tutors help when they first started learning programming (i.e. when they were Keen Beginners). However as their skills progressed to competent (Budding or Advanced Developers), having learnt how to use the debugger diagnostic tool, these students became less reliant on the tutor’s help, preferring to attempt solving their coding issue themselves.

“As you get better and better, you don’t need to go to the teacher as much. You can work out your problems yourself.” (I15)
“Using the line-by-line debugger and / or using try statements, it allows you to get through your code and identify, ‘This value’s not adding up here. That is throwing an error in that section.’” (I18)

These highly engaged students have a strong ownership of their learning and seek to solve their programming coding problems on their own. They only seek a tutor’s help for really challenging problems such as the final phase in their assignment. Some students described trying to work out a problem themselves for hours and even days, prior asking teaching staff.

“I think this comes down to just the type of person that I am. I would probably work my way through it until the next day. You have tutes once a week. My tutors were all approachable but I also had a big ego and thought, ‘I can do this and get through it on my own,’ so in that aspect I didn’t always ask for help.” (I14)

In this scenario, emailing the tutor or waiting for the next class is less appealing as the student is able to more readily make sense of the information that is posted on online forums.

“When it comes to email, I can’t wait for the reply. I’m always in rush, so I always try my best up until the point where there’s no choice but to ask.” (I12)

“I am comfortable using the forums, the response time is fast, and, it’s on mobile, so, it is really convenient as well.” (I12)
4.4.5 Summary

Personal Networks describe the different people students consult or seek information from in the process of learning programming. Personal Networks include a wide range of personal contacts beyond just the peers within the same class or year level. Personal Networks beyond the class community include friends in the same degree but at later stage in the degree, friends outside the degree studying in the same or different university and friends in industry or family. Personal networks also include teaching staff lecturers, class tutors and help lab tutors. Help lab tutors are students from the same degree. Some students will continue to consult help lab tutors outside of the help labs and even when they no longer work in the help lab – they become part of the students’ wider personal networks.

The majority of the student interactions with their Personal Networks happen outside the classroom. Even though students work on their own or consult some peers or their tutors inside the classroom, the bulk of the interactions with personal networks expand beyond the classroom. Many of these regular consultations take place face to face or via online media, such as text messaging, Facebook or email.

Sometimes students consult multiple personal networks for different levels of help and assistance. Some students described consulting debugging problems with peers within the same class, or friends that are more knowledgeable about a topic. At the same time, these students also described consulting personal contacts from their wider personal networks (friends from high school, or from other universities, or from industry or family) for how to approach and code a task.

Interactions with peers begin informally in the class room on a needs basis but most of these evolve to more regular interactions outside the classroom. Towards the end of the first course students meet more regularly as a study group where a more formal type of interaction takes place. However these study group interactions happen mostly outside the class room where members have the same skill level or some of the members have higher skill level compared to the other members.

When seeking help from staff, students tend to seek the most convenient source, depending on the type of issue, the urgency and time they have available. Asking in the tutorial or attending a help lab is preferred over making an appointment with the lecturer. Some students may feel intimidated to consult the lecturer, because of fear of judgement, especially if they are not confident with their skills. Students who are confident consulting teaching staff were those who were seeking help with advanced issues. The more advanced students will tend to try all other options before asking staff for help. Most novice students prefer face to face consultations, as they require a richer explanation to understand and visualise what is causing a coding problem and how to fix it.
4.5 Information Sources

This section describes the range of information sources that programming students discussed using in this study. All the students used both written sources and online sources. Written sources included subject (course) materials provided to the students, as well as other written materials the students may have sourced, including books.

Students reported using a very wide variety of online sources. These included the Google search engine, online manuals, online tutorials, YouTube, external programming forums and internal course forums.

All student profiles use most or all of the information sources described in this section but do so in different ways and with a different order of preference. These differences between profiles are described in Patterns of Information Use (section 5.3). The information sources preferred by each student profile are described in Table 6-3 - Profile Information Behaviour, found in section 6.4.4.

4.5.1 Online Search Engines - Google

Online search engines, and Google in particular, were used by all of the students in this study. They used Google to not only search for information to help them solve a coding or debugging problem, but also as a source to learn from. Through a Google search, students explore language specific online documentation and tutorials and also find postings on external on-line communities and forums (outside the University) where professional programmers as well as other programming students participate.
“If I was getting certain errors I would normally check my coding first; I would then normally Google word-for-word typing in sections of the error message into Google and then I would get forum posts on Stack Overflow and find someone else who is trying to do same thing. Essentially I would start with Google – typing the error into Google and working from there.” (I13)

All the students in this study described using Google as their next point of reference after looking at their notes, especially for assignment deliverables. The beginners use Google as an online manual looking for quick syntax reminders and also for sample code. The more competent students would search Google for the more advanced tasks in their assignments. Even though these students were able to understand information found on a Google search for the basic parts of their assignment, the challenge was, understanding the searched information in regards to the more complex parts of their assignment deliverables. The more advanced students use Google for learning advanced features.

“I used Google but there were many concepts that people were trying to explain which I did not understand because they were on different levels of what I was.” (I26)

Speed of response was mentioned by the more competent profiles as a reason they preferred using Google over emailing teaching staff or sending private messages over the internal forum.

“You Google it, you do your research and then you get an instant solution to the problem. It is much faster, if you use the right keywords.” (I22)

A fast response is also important to the beginners but these students generally prefer face to face consultation over doing a Google search because they cannot understand what they find.

4.5.2 Online Manuals

For most programming environments, a set of technical documentation and user guides (online manuals) can be found online that help guide users on how to use programming syntax. The information is presented with code examples and explanations that are language specific.

Approximately a third of the students in the study described using an online manual of some sort (10 of 31). The only online manual specifically named by students was MSDN (Microsoft Developer Network), used by 4 of the students (of 31). These students described using the MSDN Library to refresh their memory on how to write specific code syntax: for example how to set up the condition in a Do While or Do Until loop or to look up how to set up the parameters for a specific function call.
“Sometimes I just needed to refresh my mind of how to use an IF statement or something and just see the format. It wasn’t to understand – it was just to quickly refresh because I was already on the computer.” (I19)

They also referred to MSDN as a learning resource, to learn how to write code functionality that is part of an advanced feature or for a language that they are learning independently.

“MSDN is just a godsend if you don’t understand particular methods. I know there are websites out there that can assist you but MSDN covers everything. It can give you multiple ways to do things in a clear, decisive manner and that’s probably what I like the most.” (I18)

4.5.3 Online Tutorials

Online tutorials provide information in a more structured manner to assist users with learning independently by following the exercises for a particular programming language. Three students (out of 31) used W3Schools for web problems and two students (out of 31) used Code Academy. They described using online tutorials to look for additional examples to complement their written notes.

“If the examples I tried from the lecture notes didn’t work I would then search online and go to Code Academy or W3 schools. They had a group of tutorials about how to do arrays.” (I26)

“For HTML, JavaScript and web programming the W3 Schools was fantastic for that. They would show you examples.” (I8)

Online tutorials were also described as a way of learning new programming languages independently, that were not part of their formal studies.

“I am fairly comfortable with programming and I am confident to learn C# on my own. There are some differences and technicalities with Java so I would go to the online tutorials, W3 school website definitely helped.” (I2)
4.5.4 YouTube

Students described using YouTube videos to learn how to do specific things with programming, including structuring an algorithm that was challenging for them. The students who were more inclined to use YouTube described themselves as being visual learners and preferring to visually see how things can be done. They find it easier to watch rather than follow guidelines from other sources and find it useful that they can go back and repeat certain sections. Five students (out of 31) described using YouTube.

“I used YouTube websites a lot, but with YouTube had to know exactly what you're looking for. One time I knew how arrays worked but I was confused as to how to do a search. I went on YouTube and this lecturer from San Francisco showed how to do three different searches similar to my problem.” (I26)

“I would look at YouTube videos, uploaded from other universities which was helpful, not just for IT but for other subjects as well. Normally I found videos one of the best sources, because you can jump around and it shows you what it is doing.” (I13)

Even though YouTube is quite popular with students and some refer to it for how to code a particular problem, students also face some challenges. Getting the correct YouTube video requires knowing the correct keywords to search. Beginners will have difficulties at the start in knowing what keyword search words to use to find what they are looking for.

“With YouTube the challenge was knowing what keywords to use and also knowing exactly what you were looking for.” (I26)

4.5.5 External Forums

External forums are online communities outside the university where professional, amateur and hobby programmers as well as programming students from around the world, read and post information about programming problems and solutions. The dissemination of information is done in an unstructured way and on a needs basis. Someone who may want to solve a particular problem may do a search using Google and then find postings about the issue on a forum. They may initially read the relevant postings. If the information is not clear enough or complete, they may then post a question on the forum for someone to answer. Half of the students in the study described using external forums (15 of 31). Seven described using the Stack-Overflow forum, three used the Whirlpool forum and five students did not specify the name of the forum they used (see Table 4-12).
“Over time I got to know the Stack Overflow forum for programming, and a couple other ones. I could then recognise and I would go to them first because you could see when answers were ticked ‘Yes, this helped me.’” (I8)

<table>
<thead>
<tr>
<th>Online Forums</th>
<th>Read Only</th>
<th>Read and Post</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack Overflow</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>Whirlpool</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Unknown Forum</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 4-12 - External Forum Usage

Two thirds of the students described only reading posts from these external forums, mostly to look at how others went about solving similar programming problems. Only one third of the students described posting a question about their programming problem on an external forum.

“No, I have not posted to Stack Overflow. I just used it as a resource, not as a discussion forum.”(I29)

External forums were described as being an available and conveniently accessible information source outside of the classroom. This is particularly relevant to part-time students who tend to seek information after hours and have more difficulty developing personal networks with fellow students, due to the nature of their work and study.

Some students used external forums as a quick reference resource for accessing sample code, where they may have skipped the planning or thoughtful adaptation from code used in learning tasks. Others used external forums as a learning resource, to obtain a general idea about how to code a problem and to verify if they were on the right track. Alternatively they sought how to code advanced functionality that was beyond the scope of what they were learning.

Many of the external forum users mentioned that they had a high level of trust in the forum as a source of information. This stems from the belief that thousands of users, members of a specific forum, would have at some point answered a similar or identical problem to the one the student is facing.
“The first few times I had a problem, I would use Google to find what the issue was and I realised that so many people have the same issues as us and they are probably ex-students who posted on forums. I think I trusted it because I knew most of my problems probably will be solved there.” (I8)

“If you go to Stack Overflow, there are thousands of users to safely guess that it is going to have the correct answer there.” (I13)

The more advanced student profiles are able to strategically choose the information source which will give them the fastest response and most suitable solution to their problems, using external forums, Google and their written notes as appropriate.

4.5.6 Internal Forums

Internal forums are online communities provided by the university for currently enrolled students within a subject (course) for a semester. Two internal forums that were provided to students were Piazza and Blackboard. 17 of the 31 students did not have access to internal forums, either because the forums were not available at the time they studied programming or because the teaching staff at the time chose not to use them. Of the 14 students that did have access, half only read posts from staff and other students. The other half both read and posted openly to everyone on the internal forum (see Table 4-13).

<table>
<thead>
<tr>
<th>Online Forums</th>
<th>Type</th>
<th>Read Only</th>
<th>Read and Post</th>
<th>Provide Help</th>
<th>Private messaging</th>
</tr>
</thead>
<tbody>
<tr>
<td>Piazza</td>
<td>Internal</td>
<td>6</td>
<td>5</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Blackboard</td>
<td>Internal</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4-13 - Internal Forums

The students who read posts from other students on the internal forum described a sense of relief knowing that others were experiencing similar types of issues (9 out of 31).

“It was good to see that a lot of students had problems with the same things and it was easy enough to find the solution. The responses were so quick, that was good.” (I19)

“It makes you feel more confident if you know that everybody else is experiencing the same problem.” (I25)
Some students described checking external forums first, and only when they had exhausted this avenue, would they consider posting a question on the internal forum (6 out of 31).

“I didn’t post my things straight up in the internal forum. Instead I looked it up on Google and tried to do it myself because you don’t want to post stuff and then feel like an idiot if it’s a simple thing because you’re showing it to everyone.” (I20)

Some Reluctant and Willing Beginners did try to post questions on the internal forum but found it difficult to understand the tutors’ indications/advice when it was provided. These students found interpreting the information to help fix a coding problem difficult to follow both in the internal but also in the external forums as well.

“The internal forum was good but difficult at the same time because sometimes it was hard to know what the tutor meant and it would be like sending slabs of code or an image and asking, ‘What’s the error? How do you get that?’ It’s hard when you’re not face-to-face with the person.” (I25)

A few students described reading only information posted by teaching staff with respect to course deliverables such as clarification of specifications and due date confirmation and extensions. These students described a strong preference for using the external forums (2 out of 31).

“I would ask on the internal forum for things related to the actual requirements of an assessment, if something was worded strangely and I was confused. But I would look through forums or FAQs for technical issues.” (I22)

Of the students that didn’t post on internal forums, the most prominent reason mentioned was fear of judgement by other students and/or teaching staff (4 out of 31). The internal forum is a much smaller community compared to an external forum. In the internal forum, the lecturer, who is the moderator, knows most of the students and the students know each other as well.

“I guess I didn’t really like the internal forum because it would be slower to get a response. I guess you look a bit silly if you were posting every single problem on it especially since the problem is so easily solvable with a search engine.” (I22)

“If I was afraid to ask, then I would read and analyse responses to questions asked by other students. If I was a bit ashamed of asking for help, I would log on as anonymous.” (I27)
The students who answered posts from other students over the internal forum were mostly from the Budding and Advanced Developers (3 of 31). These students displayed not only their thorough understanding of programming issues but also the maturity not only in age but in being at a later stage in the degree. One Keen Beginner described a reluctance to provide help over the internal forum for fear of confusing or intimidating other students.

“The questions I saw on the internal forum were questions that I had already found an answer for. I didn’t help too much because when I would explain things verbally some students would say ‘That’s not what I’m doing’ and I did not want to freak everyone out.” (I13)

Four students described using the internal forum as a way of communicating privately with teaching staff as an alternative to email.

4.5.7 Written Sources

Written sources are the subject (course) materials such as lecture notes, tutorial notes and additional course materials provided by the teaching staff. They also include books and any other written materials the students may have sourced from outside their learning environment.

All 31 students in this study described looking at the lecture notes and tutorial notes (learning tasks) during their study of programming. However, not all the students processed the information that was handed to them and understood the information in the same way.

Some Reluctant and Willing Beginners described skipping some lectures and not fully understanding the lecture notes when they read them later. Some of these students also described skipping tutorials and/or not completing the tutorials till much later in the semester when they were due for final marks. These students who initially skipped reading some of tutorial notes, would have also being unable to refer to their tutorial notes while they were working on assignments (deliverables) as they would have worked on assignments with incomplete or misunderstood learning tasks.

“I struggled to understand from the notes even though a lot of people found them easy to understand. I suppose not going to the lecture made things a lot harder to understand because I wasn’t getting that the explanations that everyone else was. I did look at the additional complementary course notes, but I didn’t go through them in detail.” (I21)
Some beginners start off with the belief that the lecture notes are for the exam and do not refer to these while working on their tutorials (learning tasks) until much later on when they become acquainted with the system and also learn from other students.

“During first year, I did not check or looked back to the lecture notes as much because we were quite new to the whole system of being at university. Now that I am in second year I have learnt from that mistake and now each time I have tutorials, I always have the lecture notes on the side. I do split screens, Google running on one side, and the tutorial worksheet or lecture notes on the other.” (I27)

The Keen Beginners and Budding Managers described actually not just looking at the notes but also trying out the sample code from their course notes prior to searching online for sample code.

“The first thing I would do is go over the lecture notes and the additional complementary course notes the lecturer gave us because they had the examples of code for different things. I would try their method and if it worked and I was happy, I would use it. If they didn’t work I would look at W3school or YouTube but that was another huge challenge.” (I26)

The more highly competent and advanced profiles (the Budding Practitioners, Budding Developers and Advanced Developers) described referring to their tutorial notes and completed tutorial code when working on their assignment.

“In the intro course, what was in front of you was relevant to the task at hand. So as long as you knew that information, it was easily adaptable to the final product. That was the beauty of it!” (I7)

Furthermore, they described going back to their tutorial notes as a point of reference, but also when their assignment code had issues, to check if they had missed something.

“I picked up loops. Every time I needed to introduce a loop, I would look back on a tutorial where we did a similar task but using three different loops. I looked at the way the loop was written and from that I could instantly remember how to use that loop.” (I22)
4.5.8 Summary

Students in this study used a wide variety of information sources, both written and online. A very wide variety of online sources were used, including the Google search engine, online manuals, online tutorials, YouTube, external programming forums and internal course forums. Written sources included materials provided by teaching staff (lecture, tutorial and course notes) as well as books.

All students described using the Google search engine but they use it in widely different ways. Novice students tend to use Google as an online syntax reference and to search for sample code, while the more competent students search for information on more advanced features. To many students, Google is preferred as a problem solving tool because of its speed of response compared to emailing teaching staff.

Online manuals, such as MSDN, provide students a reference resource with which to look up the syntax of code they are trying to write, as well as a way to learning advanced programming constructs independently. Online tutorials are used in a similar fashion, looking for additional examples to their notes and to learn advanced concepts. Some students also use online tutorials to learn other programming languages independently. YouTube videos were described as a preferred source by students who saw themselves as visual learners. They find it easier to watch rather than follow instructions.

External forums are online communities where questions and answers about coding issues are posted and discussed by programmers world-wide. While all the students using external forums read posts, only one third posted questions. They are a convenient information source available outside classroom hours, especially for part-time students. External forums are used as a quick reference to look for sample code, as a learning resource and as a guide for how to code advanced features. Students who use external forums place a high level of trust on the information they find, based on the belief that thousands of users on a forum would have likely faced a similar problem.

Internal forums are online communities only available to the students in the same course during their study. Most students who only read posts described feeling a sense of relief knowing that other students faced similar issues. Some of the beginners, however, found interpreting the help that is posted difficult to follow. A number of students described being reluctant to post for fear of judgement, mainly from other students. The students who were confident enough to post questions and answers tended to be the more advanced and mature students. These students tend to consult external forums first before posting on the internal forum.
Written sources are materials provided to the students, such as lecture and tutorial notes. All students described referring to their written notes. Some students, however, have difficulty using the notes because they begin working on assignments with incomplete or misunderstood learning tasks. Other students read and try out the coding examples in their written notes before looking online. The more advanced students described referring back to their written materials and completed tutorial work for their assignment.

The more advanced students are able to strategically choose the information source that will give them the fastest response and most suitable solution to their problems. These students initially use Google but rapidly learn which online sources will help them with the additional exploration they are after.

4.6 Chapter Summary

This chapter introduced the Theory of Influences on the Student Learning Experience of Programming. This theory was developed from a qualitative study that interviewed 31 students. The main model was presented and briefly outlined. The outer elements of the theory were discussed in detail: Student Characteristics, Personal Networks and Information Sources.

Chapter 5 continues the presentation of the theory and describes, in detail, the Student Learning Behaviours component of the model: Core Learning Perspectives, Patterns of Collaboration (including the Model of Dependency) and Patterns of Information Use.
Chapter 5  Student Learning Behaviours

Chapter 4 introduced the central model of the Theory of Influences on the Student Learning Experience of Programming. This model brings together the many influences that arose from the analysis and presents them in a unified and cohesive way. Central to the model is the concept of Programming Learner Profiles, which encapsulates key influences that define and describe different types of students and how they go about learning to program. Learner profiles are defined by the concepts of Perceived Personal Relevance, Learning Trait and Skill level. The Core Learning Perspectives and Behaviours described by the profiles include the student’s Ownership of Learning, Learning Task Intent and Problem Solving Behaviours. Patterns of Collaboration describe the various levels of dependency students have with their Personal Networks. Patterns of Information Use describe the various ways students use the Information Sources available to them.

Chapter 4 discussed, in depth, the foundational elements of the model: Student Characteristics (General and Programming Context), Personal Networks and Information Sources. This chapter continues the in depth discussion of the model. It focusses on the Learning Behaviours component of the Programming Learner Profiles.

All students have personal networks and information sources available to them but use them in vastly different ways. This chapter firstly explores the concepts within Core Learning Perspective, which include Ownership of Learning, Learning Task Intent and the many Problem Solving Behaviours students practise. Then the chapter explores Patterns of Collaboration, which includes the Model of Dependency that was developed to show the levels of dependency inherent in the way different students interact with their Personal Networks. Lastly this chapter explores Patterns of Information Use, which describes the ways in which students use the various Information Sources available to them.
5.1 Core Learning Perspective

Core Learning Perspectives describes the key attitudes, intentions and behaviours of students as they proceed with their learning of programming. Core Learning Perspectives brings together three concepts: Ownership of Learning, Learning Task Intent and Problem Solving Behaviours. These concepts were found to be key influences on how students collaborate with their peers and personal networks and use their information sources.

The Ownership of Learning concept describes the students’ attitude towards what degree of responsibility they take for learning programming. Students with a strong ownership take responsibility for their own learning while students with a weak ownership take the view that they are there to be taught.

The Learning Task Intent concept describes the goal or intention students have when doing their learning and deliverable tasks (assignments). Students with a strong Learning Task Intent have a deep desire to understand the code they are learning while students with a weak Learning Task Intent want to simply get the task done to get the marks and move on.

The Problem Solving Behaviours concept describes the various steps and actions students undertake when solving programming problems during their learning.

Ownership of Learning, Learning Task Intent and Problem Solving Behaviours are closely related and intertwined. In general, students who have a strong Ownership of Learning also have a strong Learning Task Intent and practice thorough Problem Solving Behaviours. Similarly, students with a weak Ownership of Learning tend to also have a weak Learning Task Intent and Problem Solving Behaviours. This is not a hard rule, however, and some students may have a mix of both strong and weak aspects of these three perspectives.
The students’ attitudes about their learning responsibility influences what will gear their intentions towards their programming learning tasks - whether they want to conceptually understand a problem or just get the task done. These intentions will in turn influence what the students do and how they approach it: their behaviours, actions and steps they take when they problem solve.

In some cases, students may have mixed Ownership of Learning, Learning Task Intent and Problem Solving skills. Novice students that are still developing their skills may have weaker Problem Solving Behaviours but strong Ownership of Learning and Learning Task Intent. Other students may have a will to learn and have a strong Ownership of Learning but may be constrained by their circumstances and not have enough time to devote to their learning tasks and develop their problem solving skills.

### 5.1.1 Ownership of Learning

Ownership of Learning describes the students’ attitude towards what degree they assume their responsibility for learning programming. At one end of the spectrum, students with a strong ownership take responsibility for their learning. These students need to do things themselves and ensure they understand. At the other extreme, students with a weak ownership take the view that they are there to be taught and it is the teacher’s responsibility to ensure they understand. Ownership encompasses the following: how much they immerse themselves in the problem they are about to solve, how much time and effort they put into trying to understand the problem themselves, and how deeply they look at their learning materials at hand (written or online).

Students who display a strong ownership seek to work out a problem themselves when they learn programming. They do extensive practice of their learning tasks to seek understanding. They are keen to learn to solve coding problems independently so they spend extended amount of time, consulting multiple sources to achieve this goal. They consult others as a last resort. Their interactions with their peers and personal networks are collaborative where they learn by critically comparing approaches. They also quite rapidly adopt coding tips, debugging techniques that they learn by observing how more experienced peers, personal networks and teaching staff use these.

Students with a strong Ownership of Learning have a deep desire to understand the programming concepts they are learning. They treat coding problems as personal challenges that they want to work out on their own.
“I want to work it out for myself. I don’t want someone to give me the answer straight up.” (I29)

“So I reckon, based on my way of looking at my problems, I try to sort it out myself as much as I can.” (I15)

In this quest, they also persevere for a longer period of time in order to fix a coding problem. They described experiencing a strong satisfaction when fixing a problem themselves even though they may have also experienced frustration along the way.

“I was very adamant about solving the problem rather than leaving it and coming back to it. I just kept at it. There were times I went to sleep at 6 am because I felt obsessed about solving the problem - but I really enjoyed programming.” (I3)

In the strategies and steps they describe, they emphasize the importance of practice in their learning. They learn by practicing their learning tasks (tutorials) mindfully. They re-do key parts of a learning task without following the strict step by step guidance. This helps them immerse themselves deeper into the problem to gain a better understanding.

“When I was not able to finish a tute or I didn’t understand how it was working, I would step through it all with the debugger. I’d also hand write it - up to ten times. I would work out exactly how it was working. Then in my assignment, I could manipulate it into a more generic way. You’re never going to fully understand it until you actually step through it all, and see how it’s actually working.” (I15)

“I did every tutorial and if I didn’t understand it, or it didn’t really make as much sense to me I’d just do it again. Especially when I was studying for the exams, I did them all again.” (I15)

These students also learn from the mistakes they come across as a result of the additional practice they do. These new mistakes trigger additional facets that were previously unexplored as they try to work out why their code does not work as expected.

“I find you learn more from having problems! If everything goes smooth from the get go, you don’t learn anything whereas once you’re finding there are bugs or issues, you spend the time to find that out and you learn more.” (I7)
These students also tend to exhaust all avenues in trying to solve a problem prior to seeking any assistance, if any.

“I know I had worked on it and I had tried to do it myself. I knew when I had to ask a question it was legitimate. It wasn’t something I was doing because it was the easy option out. It was because I didn’t understand it and I needed to consolidate.” (I18)

“Before I asked for help I tried to understand it first. I tried to figure out by myself and do a simple example. I would use the debugger first, and then I would try to Google. I tried to understand the logic of how it was implemented and the result it would give.” (I12)

Independent learning is also another characteristic of the learning responsibility they take. When they seek assistance, whether from online external forums or from their peers or teaching staff, they proceed to try out themselves the various suggestions in their own code. They come to a better understanding by comparing the implementations of the different ideas of how to go about coding or fixing a problem.

“Often I would talk to my friends and sometimes my tutor when I was really stuck. I would give it three or four tries, using three or four different methods. I tried to do the suggestions. I used the template of my code and just implemented the suggestions using different routes… so if one didn’t work I would try the next plan.” (I2)

“I was always looking at Google because it was a learning curve as well for me. Even if I didn’t have an issue, I was curious to see what other things are out there because I wanted to improve what I was doing. To make it more simplistic.” (I15)

In this study, 11 (out of 31) students displayed a strong ownership of their programming learning.

Students with a weak ownership believe others are responsible for their learning. They take the view that they are there to be taught and it is the teacher’s responsibility to ensure they understand. They assume a minimal level of responsibility and often rely on others for their learning of programming. They lack the strategies and know-how for dealing with coding problems but also, most are not interested in assimilating the strategies that can help them solve their coding problems independently or they just find these techniques difficult and time consuming.

“I want to be able to come to school and feel that I will learn something, that the teacher is going to teach me something, not me going off and learning by myself. Sometimes, me and my friend felt like we were left in a dead end.” (I4)
Their weak ownership is illustrated by various behaviours such as “fast-tracking” techniques where they take short cuts with their learning tasks (tutorials). For example, they partially complete their learning tasks in class with help, or skip them altogether. They do not follow up on the learning tasks that were misunderstood, or they did not know how to complete. Instead, their preference is to work on those tasks that earn the marks (i.e. the assignment deliverables). They begin working on the assignment deliverables with incomplete and/or misunderstood learning tasks.

“I could do some of the learning task (tutorial), but then not all of it would be complete by the end. Whether you learn that before the assignment or during, it’s just that knock-on effect.” (I10)

Unlike the students with a strong ownership, those who have a weak ownership do not engage in mindful practice where they redo parts of their learning tasks from scratch in order to understand them. They acknowledge that practice is required in order to understand but they don’t spend the time practicing.

“It really comes back to applying more time than what I was able to. You need to practice in the same way when you need to use new sentences or new words when you learn a foreign language. No, I didn’t do that extra practice. I didn’t feel as though I had the time to catch up and do it.” (I9)

They also show dependent strategies such as using sources of help and or personal networks for coding and debugging problems. In this case, their understanding is compromised because their approach is quite passive.

“The focus was really just to get the tasks done and get the points that were allocated for that week’s tutorial. Not everyone understood really. I think the people who enjoyed it would go back and read the prescribed text and the chapters to try and make sense of it.” (I11)

They consult the sources of help or use their personal networks, however they take the help and explanations in a passive way. They receive the help and then they move on to the next thing even if their understanding of the area they received help in is still vague. They don’t retry the task again, or they don’t re-do the bit of the task where they received help in order to understand why something was not working.
“On my own, I would probably sit there for a good five minutes, just trialling and error, cutting everything out and retyping it again. But after a while it gets frustrating. I did not really use the debugging tool - I just saw how people did it. I didn’t really analyse it as much. When there was something that was highlighted in the programming environment, if I didn’t know what it was referring to, my tutor would give me a clue, but if I didn’t get it he would then tell me.” (I27)

In this study, 10 (out of 31) students displayed a weak ownership of their programming learning.

An intermediate ownership, on the other hand, describes a mixed learning responsibility. There is variation in the level of responsibility students assume. They genuinely want to understand and work out the problems that they see relevant. They are proactive in assuming responsibility for learning they perceive is relevant, but their ownership weakens and they become co-dependent on others in areas that they do not see is relevant to them.

“I spent the time to get it working and understand the basics. I did not look at using files as it is not really important for me at this stage. I’m not going to do programming so I had to think of the cost benefit.” (I31)

In their proactive phase, they do try the suggestions found on online forums prior to asking teaching staff. Some of these students treat teaching staff as a last resort.

“Notes were my first point, Google was my second point and tutors were my last point. I did not want to waste the tutor’s time if I could figure it out by myself. Especially if it’s something I can just find out. It was more about the apprehension of looking like an idiot in front of the tutor.” (I20)

In their passive phase, they seek help and rely on others primarily for debugging coding problems. This is the phase they tend to delegate and rely most on others but they also seek help from others for how to get started with coding.

“When I had a problem I usually tried to definitely work it out by myself first so from textbooks or the Internet. I then would ask people in my class. Outside the tutorial I had a friend that tutored me if I had some really big issue that I couldn’t overcome so he was sort of my ‘go to person’. I also would go to the help Labs and seek help.” (I16)
There are many possible variations of intermediate ownership, depending on the individual student’s circumstances. Some part-time students, for example, are forced by the nature of their study load to work independently, but are not interested in learning programming as they do not see it relevant. There are also some students who have the desire to understand but do not have the time to study for different reasons so they learn by relying on others. There are also those students who clearly have a strong desire to understand and are in the process of assuming a stronger ownership of their learning

“I try to always work it out myself and learn as much as I can within the time period but know when enough is enough and to move on.” (I18)

“I came into the course thinking wow we are going to learn something but it is just too hard and I haven’t understood the basics yet. I tell you, I’ve worked nearly all my life and I’ve never had so much stress coming at me going into the lecture and going into the tutorial. I’m frustrated with myself with why I cannot understand it. When I started the course, when I did the tutorial, I actually went home and spent two hours doing the whole thing over again but now it is just too much work every week.” (I4)

In this study, 10 (out of 31) students displayed an intermediate ownership of their programming learning. For example, 6 (out of the 10) students discussed proactively trying to work out coding problems themselves by looking through their notes for sample code but also actively seeking online help prior to seeking help from their sources of information or personal networks. They discussed consulting online (external forums) for samples of code as a guide for how to code or how to fix coding problems but not all of these students found they could implement what was suggested in the forums. 4 (out of the 10) students described a more passive style, while taking responsibility, also behaving co-dependently with peers, personal networks and teaching staff.

5.1.2 Learning Task Intent

Learning Task Intent describes the students’ goal or intention they have when doing their learning tasks and deliverable tasks (assignments). Students with a strong Learning Task Intent have a deep desire to understand the code they are implementing while students with a weak Learning Task Intent want to simply get the task done to get the marks and move on. This may be for the whole or part of either the learning task or deliverable.
This dimension is closely linked to their motivation to learn programming. Students who perceive that learning programming is relevant to them for various reasons described having a deep desire to thoroughly understand their learning task or deliverable. They are intrinsically interested in learning it so they willingly do their own exploration in order to seek understanding which surpasses understanding to pass examinations. The deep question they ask themselves that drives what they do is ‘do I understand this so I can use what I am learning in the future’? On the other hand, the students who do not perceive programming as relevant to them, who are extrinsically interested, described their Learning Task Intent as “getting it done” and out of the way to achieve the marks and move on. Their seeking of understanding is to pass examinations.

Students that display a strong Learning Task Intent have a powerful interest to understand how the code they have come across works. This may be code implemented by themselves or others. They also strive to understand the problems they have encountered in the coding, what triggered these problems, how they were fixed and why the fix worked.

“Well I guess that was a key moment I remember where I sort looked at my friend’s code and I could say, ‘Ah, I can see your problem now.’ and then it sort of helped even my own code because I’m like, ‘Okay, I’m getting now where things should be.’ I had that attitude that I want to know why I don’t understand something but not all of my friends are like that.” (I8)

The students with a weak Learning Task Intent are those students who perceive very little personal relevance in learning programming. They mostly have an extrinsic interest such as a general interest in learning IT or IS or just acquiring a degree in these areas. Their learning intent for programming is just getting the task done to get the marks. These students are the ones who display various types of short cuts with the least understanding.

“People who were not motivated, like me, we were just focused on looking and making sure we completed the tasks and got the marks. You were just trying to get something to a standard which was just working, as opposed to trying to get something that was excellent. I think it might have been just me, just not enjoying it and not seeing the benefit in completing a task to its best potential.” (I11)
Students with an intermediate Learning Task Intent have a desire to understand what they see is relevant. They may perceive some of the programming topics relevant to learn but not all, or they may perceive some of the programming phases relevant such as laying out of the Graphical user interface but not the coding itself nor the debugging process. For any learning that is strictly outside the boundaries of relevance, their desire to understand subsides giving way to extrinsic reasons such as the desire to get the task done and or pass examinations. Usually this strategic manoeuvring involves taking short cuts that will inevitably leave gaps in their understanding.

“I did OK in the subject however I did not really get to understand it as fully as I thought I might have. I love information technology, but programming is just one of those things that I suppose you just got to keep working on.” (I21)

“I remember when I was hating programming I was really hating the situation I got myself into for not dealing with it or not managing as I went along. It was very frustrating. Though I did have breakthrough moments where you have been busting on something and then it works and you understand why it works or even if you don’t understand why it works it still works and that is a really good feeling.” (I9)

5.1.3 Problem Solving Behaviours

The Problem Solving Behaviours concept describes the way students approach solving coding problems. What they do and how they do it, the behaviours, actions and steps they take to solve a coding problem. These involve the planning they may do, the way they approach the coding and debugging. The way students problem solve coding problems is influenced by their Ownership of Learning and Learning Task Intent.

Ownership of learning, Learning Task Intent and Problem Solving Behaviours are closely related and intertwined. In general, it follows that students who have a strong Ownership and Learning Task Intent have a strong problem solving approach. However, in some cases, the way students problem solve is developing so it is not as strong as their Ownership and Learning Task Intent. It lags behind. Similarly, students with a weak Ownership and Learning Task Intent have a weak problem solving approach. What they do and how they problem solve is not conducive to developing independent problem solving skills.
The Problem Solving Behaviours described by the students in the study were classified into three groups: thorough, intermediate and ad-hoc (refer to Table 5-1). The rest of this section provides a brief description of each problem solving behaviour.

<table>
<thead>
<tr>
<th>Thorough</th>
<th>Intermediate</th>
<th>Ad-Hoc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thoughtful Planning</td>
<td>Simple Planning</td>
<td>Lack of Planning</td>
</tr>
<tr>
<td>Thoughtful Error Detecting</td>
<td>Thoughtful Trial and Error</td>
<td>Ad-hoc Trial and Error</td>
</tr>
<tr>
<td>Desk Checking</td>
<td>Coding Step by Step</td>
<td>Unsystematic Approach</td>
</tr>
<tr>
<td>Efficient and Independent Debugging</td>
<td>Developing Independent Debugging</td>
<td>Dependent Debugging</td>
</tr>
<tr>
<td></td>
<td>Relying on sample code</td>
<td>Coding in Big Chunks</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Learning Task Skipping</td>
</tr>
</tbody>
</table>

Table 5-1 - Problem Solving Behaviours

**Thorough Problem Solving Behaviours**

Students that display thorough problem solving skills in programming, approach each programming phase (i.e. planning, coding, debugging) with thoroughness and pro-activeness. They do thoughtful planning, usually spending time thinking about what needs to be done and how. They are more practiced at structuring an algorithm. They do extensive practice of their learning tasks to seek understanding.

They are keen to learn to solve coding problems independently so they quite rapidly adopt debugging techniques learnt by observing more experienced peers, personal networks and teaching staff use. They interact collaboratively with the peers and personal networks with the intention of seeking a deeper understanding.
Thoughtful Planning

These students usually spend more time thinking about what needs to be done and how, prior to starting their coding or making changes to their code. They may choose to write it down in a more structured manner such as using plain English statements or they may do it in an unstructured fashion such as drawing up a plan on paper regarding what variables they may use, order of events, places in the code where variables will be changed and structure of the constructs.

“What I’ve found that has helped a lot was writing down exactly in plain English, step-by-step what the code needs to do. Something like pseudo-code. I wrote it down and then the next day when I came back to it fresh, I had it working pretty much in five minutes’ later.” (I29)

“Working with Assignment 3, I realised that I needed to start again. I drew everything out on paper, the variables I’d be storing and what I’d be doing and how it would all relate to each other. So I started again with a better plan and a different headspace.” (I13)

8 (out of 31) students in this study described making thorough written plans prior commencing their code.

Thoughtful Error Detecting

They are also faster at detecting syntactical, semantic and logical code problems. Their additional practice enables them to come across more problems, building their knowledge base of the different types of errors that can be encountered in the code and how to solve these. They also display proficiency in the techniques they use to trace through the code as it is executing, to detect where errors are occurring or where the code is not behaving as expected.

“I was using message boxes to trace the loop variables. And so when problem-solving the bug, I had to check all the loops really thoroughly and actually see everything on the line it was currently reading, and then the next line.” (I13)
Desk Checking

Some of these students display proficient desk checking techniques where they can follow how each code line will execute and how variables will change accordingly on a piece of paper. This tracing through code helps them conceptualise what events are happening as each code line executes enabling them to make simple predictions of what is going to happen if something is changed in the code. As they practice these steps they develop confidence in anticipating what will happen. Only 2 students in the study described using desk checking techniques.

Efficient and Independent Debugging

These students are also more receptive to learning faster and more efficient debugging techniques as well as being more proactive in practicing these. Usually they learn by observing more experienced peers, personal networks and teaching staff. 10 (out of 31) students in this study described practicing independent debugging using various techniques. 3 out of these 10 students described using the programming environment debugging tool.

“When errors were more difficult to identify, it was then my tutor showed me how to use the line-by-line debugger. I also learnt how to use try-catch statements in order get through code lines throwing an error. This helped with debugging because the code continues without crashing. That I think is an important debugging tool in itself.” (I18)

Seven students out of the 10 who practiced independent debugging described using other debugging techniques without using the debugging tool. These techniques included the use of pop up message-boxes or printing to the console to display values of variables and trace the program execution. Another technique involved isolating code either by separating the code to be explored or by commenting out code that did not need to be examined. These techniques were either used in isolation or in combination.

“I did play around with it or isolate it and I put pop-up boxes to try and understand exactly where my break-point was. There were many different ways that I would try and narrow it down, as opposed to just asking for help.” (I14)

12 (out of 31) students in this study described using thorough problem solving techniques.
Intermediate Problem Solving Behaviours

Students with intermediate problem solving skills exhibit behaviours that are somewhere in-between the two extremes. Those students who perceive a low relevance in learning programming described being confident and more thorough with simple coding problems but struggling to grasp how to implement and fix more complex problems. Their problem solving skills develop to the degree they see learning how to solve these problems being relevant to them. They tend to be more passive in the way they approach complex problems and often rely on others as they find these time consuming.

"Honestly, sometimes if I had no idea of how to do it I would seek for help for the entire exact thing that I needed to do, but other times, if I knew how to write a loop or what I needed to do, I would try to find an example. These were the best.” (I16)

Other students are at an intermediate stage because they are novices. Their skills are evolving as they actively practice and learn through their coding mistakes. These more proactive students begin to behave as code detectives. They seek sources of help but they themselves are becoming more proactive in the process of working out a coding problem by learning more actively from their peers. They are more willing to combine searching online forums for how others solve a similar error but also they are becoming more confident in trying simple debugging techniques.

Simple Planning

They make simple plans such as using the User Interface Design as a guide to where start with coding and/or using the code of an earlier assignment as a guide to coding subsequent assignments. Students who initially struggled with code writing described using this latter approach after some exposure and practice to writing code.

"Once I build the user interface form, with the controls, it then becomes easier. I see a button and I think I have to write code to go behind it. Once you know there is a search button, then you know you have to write the code for the search functionality and you start writing it out." (I15)
**Thoughtful Trial and Error**

They start to isolate chunks of code in separate areas in order to understand what the problem is and how it works. Few tend to delete problem code and start again.

“I thought I could put the split field outside of the loop but that did not work. It was a lot of trial and error and I know Google will not find that stuff for me. So I played around with it and then I eventually put the split into the while loop and, yeah it worked but that also took a bit of time.” (I5)

“When I started coding, if I got stuck, I would keep trying different variables. I kept experimenting with lots of trial and error. I also commented out code and used message boxes a lot to pop up what the variable is showing instead of debugging.” (I3)

10 (out of 31) students in this study described an intermediate approach in the way they go about solving coding problems of which 7 (out of 31) were more proactive in following the above mentioned techniques.

**Coding Step by Step**

They described developing the understanding that programming needs to be approached step by step, one thing or problem at time in isolation, checking that each step works prior to moving on to the next. They show an improvement of problems solving skills.

“Once I got into programming, I found it is a lot like Maths, it goes step by step. If this happens then, the result of this is x else etc. I find the basic intro OK. I can read the code and I understand it.” (I5)

**Developing Independent Debugging**

Students who begin to develop their debugging techniques described adopting the techniques they observed and learnt from their sources of information or personal networks. They compare their implemented code more carefully to code in the notes, checking for typos or missed code lines.
“I would look to see if any parameters are causing the problem with the function. I would then look at the lecture notes and course notes to try and find out what I did wrong. Specifically, comparing my code to the sample code. If it gets really annoying, I would isolate the function to look at what is inside it, but isolating takes a lot of time, especially where you have to have inputs for it”. (I20)

They search online forums looking at how others have fixed a similar problem and comparing their code to what they find. They show signs of being able to sift more quickly through the searches to find specifically what they are looking for. They understand that what they find in a Google search is only going to explain the meaning of the error that the system displays at them.

“Well I would be Googling the various commands. I would be putting in some Java code to understand what the error was to get an understanding what causes the error. Sometimes it is not clear to understand what caused the error.” (I9)

They are better equipped to look at their code and to try to find what triggered the error by using simple techniques such as pop-up messages to trace and follow where the code is working and where it is breaking down. They still seek sources of help but they themselves are becoming more proactive in the process of working out a coding problem.

“I did use message boxes to display what you think it should be or to check when it gets to a certain point and it doesn't display what is expected more so than the commenting out code. The debugger was mentioned but for the particular assignment I did not get around to utilising it.” (I9)

7 (out of 31) students in this study described developing independent skills in the way they go about writing code but mainly with debugging (i.e. fixing syntactical and semantic coding errors).

**Relying on Sample Code**

They look for sample code for how to do things more actively. They do look at their notes for sample code but this requires them to do some code adaptation. For some students this is perceived as too time consuming. For these students, searching online forums for code samples that performs the task they need to do is perceived to be a more efficient use of their time.

“I guess definitely one of the big skills that I picked up is being able to just Google something and try to find out how someone else has done that sort of thing before and I use that a lot now in every aspect.” (I16)
**Ad-Hoc Problem Solving Behaviours**

Students who display weak or ad-hoc problem solving, skip planning and jump straight into coding. They skip the preliminary planning they need to do prior to attempting some or most of the phases involved in solving a coding problem. They are unsystematic in the way they approach their learning tasks and deliverables. They start working on deliverables with incomplete and/or misunderstood learning tasks. This tendency renders them unable to recognise the patterns laid out in the learning tasks that are relevant for their deliverables and instead they go hunting through online forums for sample code. They approach coding and debugging in an ad-hoc fashion mostly practicing trial and error. They rely on others for how to get started with coding and also for fixing coding problems. For those students who see a low personal relevance to learning programming, their problem solving will remain weak for longer as they are reluctant to learn from observations. These students are not interested in learning how to solve coding problems independently.

**Lack of Planning**

These students find planning the programming task difficult so they skip this phase. The planning phase is where they put together the programming constructs (i.e. algorithm structure) that will then need to be translated into programming statements. For the novice student in particular, this preliminary plan or pseudo like English description is hard because they have just started learning this “new way of speaking” and also because it involves following a restrictive set of instructions (constructs) to carry out a task.

“Usually if there’s a sample program, I can modify it on my own. I didn’t plan because I can only visualise causal relationships, so the only possible way for me is to do it first and then find out what are the causes and then do trial and error. That’s how I usually do it because I really cannot plan. I have trouble with planning.” (I31)

“Sometimes I would not even know where to start even because sometimes our notes would go so far and if there is something we haven’t done and there were no notes for it. I know that if you know how to do this part, you should logically know how to do the next part, but I just did not even know where to start.” (I16)
**Ad-Hoc Trial and Error**

They generally use an ad-hoc Trial and Error approach to fix their syntactic and semantic (logical) problems in their code. They change code lines hoping that the code fix will work without having interpreted correctly what the environment is indicating is wrong, nor having really thought out the problem.

“I would look at it and still not know what was wrong with it. Then I just tried and put an extra semicolon at the end. If that did not work, I would try changing upper and lower case and you just keep doing that. I didn’t really get it anyway, so the only thing I could do was just trial and error and hope that something different happened.” (I11)

**Unsystematic Approach**

They also lack a systematic approach when they begin their coding phase, especially with their assignment deliverables. Students with weak problem solving skills will miss replicating the step by step process that was provided to them in the learning tasks for them to practice doing when they do the deliverables.

“I guess I am more of an interactive learner. I pretty much just start coding straight away. I think that was probably another setback, yeah! I just try things and then try different things. If it doesn’t work then I try another thing. Obviously after a while I would be very frustrated and then I decide I need a break.” (I21)

“This is coming from someone who is struggling with the code a lot, so the textbook might be useful for the tutorial exercises but when you’re doing your assignments and you get an error code, you look at the book or you look at Google and you can’t make sense. You can’t connect the two, so you trial and error just guessing.” (I11)
Dependent Debugging

Debugging a coding problem tends to be one of the hardest aspects for these students. To compensate, their debugging practice is mostly dependent on others, usually their peers and teaching staff in class and their personal networks and/or help labs outside class. They do this repeatedly without attempting to adopt even some of the basic techniques that are demonstrated to them.

“I still wasn’t very good at debugging by myself. I understood when my help lab tutor was helping me do it. If it was in front of me now I would not know how to debug it. It made sense at the time but I wish I had been comfortable doing it myself.” (I16)

For these students, learning how to code and debug remains weak or in its very early stages or may progress very slowly. They partially understand the (help) indications of how to fix a coding problem, but not fully. They show minimal progress in learning from their observations of how the sources of help consulted went about debugging so as to then practice independently how to identify coding problems and fix them.

“I usually had someone there debugging with me because to start off with I had no idea what debugging was. And even though someone had explained to me over and over again, until you practise it over and over again, I didn’t know it. I’m never one of those people that just gets it first off.” (I25)

The types of debugging described by these students consist mostly of quickly identifying typos of syntactical errors by comparing the code to their notes or to sample code in Google. For more complex problems, students tend to consult teaching staff or their personal networks.

“If it was just a little syntax error, I would look at the code for something that I had mistyped. I would then look at Google and go through the code comparing it to what I had searched on Google. As I was sitting next to a friend in the class, I would ask him if they had seen the error or if they knew how to fix it. If it was still an issue I would obviously speak to the tutor. I didn’t really try using the debugger cause I struggled understanding what it was trying to say most of the time.” (I21)
Coding in Big Chunks

Some students tend to write their code in big chunks without checking whether each step works prior moving on to the next. It is when they compile their program that they discover numerous compilation errors. They find these compilation errors very difficult to fix independently as they are unable to isolate each part or step in order to try and fix it and be able to work out whether the code fix worked.

“When you type it all in, it looks right, and you think ‘Ok, that’s going to work’ and then you get frustrated because you don’t know why it’s not working. Even still now, I can sit on a problem for half an hour and it turns out that it was the smallest little thing.” (I25)

Learning Task Skipping

Some students with weak problem solving skills begin working on their assignment deliverables without having completed or fully having understood their learning tasks.

“I was attempting to cut a corner by saving time and using as much of my content from the previous year. In doing this I created my own issues. The issue was that I could not edit any of the existing data in the database. The code that was given to us was not working and so I spoke to the tutor and he basically pointed me to one of the tutorials and asked me to work my way through the tutorial to fix the problem. I did this and it worked!” (I9)
5.1.4 Summary

Core Learning Perspectives describes the key attitudes, intentions and behaviours of students as they proceed with their learning of programming. Core Learning Perspectives brings together three concepts: Ownership of Learning, Learning Task Intent and Problem Solving Behaviours. These concepts were found to be key influences on how students collaborate with their peers and personal networks and use their information sources.

Ownership of Learning describes the students’ views on who is responsible for their learning and understanding – whether they take responsibility for their learning or assume others are responsible for teaching them. Students who display a strong ownership of their learning strongly desire to work out their problems themselves. They solve problems independently, persevering on problems for a longer period of time and consulting multiple sources, both written and online. They do additional practice and revision of their learning tasks outside of the classroom. When they consult others they do so collaboratively.

Students who display a weak Ownership of Learning believe others are responsible for their learning and understanding and that they are there to be taught. They lack the know-how for solving programming problems but at the same time are disinterested in learning the strategies that can help them solve problems independently. They take short cuts and rely on others when solving programming problems. They interact dependently with their peers, personal networks or teaching staff passively.

Students who display an intermediate Ownership of Learning display mixed views about their learning responsibility. They try to proactively work out problems themselves for the programming phases or topics they see as relevant to them, but delegate responsibility for the topics they are not interested in or find difficult. Variations of intermediate ownership include part-time students who are forced to work independently because of their study situation and also students who want to work things out themselves but do not have the time due to personal commitments.

Learning Task Intent describes the students’ intentions towards their course deliverables (learning tasks and assignments). Whether they want to understand them or just get them done. A strong Learning Task Intent describes the intention to gain deep understanding by doing further explorations than just completing the tasks to a good standard. A weak Learning Task Intent describes the intention to complete the tasks in order to get the task done.
An intermediate Learning Task Intent describes the intention to understand only those aspects of the task perceived to be relevant to the student. Some students might start with a strong learning task intent but this may weaken when the tasks get too difficult, whereas others might be interested in learning some phases of development, such as designing the user interface, but not others, such as debugging.

Problem Solving Behaviours describes the way students go about solving programming problems. The behaviours they manifest, and the actions and steps they take to solve a coding problem. What they manifest and do is in relation to how they approach the planning, coding and debugging. The Problem Solving Behaviours displayed by the students in this study were grouped as thorough, intermediate and ad-hoc.

Students who display thorough Problem Solving Behaviours approach each programming phase with thoroughness and pro-activeness. They practice thoughtful planning, spending more time thinking about what needs to be done and how prior to commencing coding. They practice thoughtful error detecting for syntactical (simple typos), semantic and logical code problems. Some practice desk checking techniques. They use efficient and independent debugging techniques and proficiently use the debugging tool to identify errors in the code.

Students with intermediate Problem Solving Behaviours exhibit mixed ad-hoc with thorough behaviours depending on their Perceived Personal Relevance and Skill Level. They make simple plans, using the Graphical User Interface (GUI) as a guide or previously written code. They recognise and practice coding systematically, working on one section at a time. These students are in the process of developing independent debugging techniques by adopting simple methods such as using message boxes. They prefer to look for sample code more actively to minimise doing code adaptation.

Students who display Ad-hoc Problem Solving Behaviours tend to randomly practice or skip some phases in the programming lifecycle. They skip planning, doing no preliminary thinking about how they will structure the program. They make ad-hoc changes to code, trying to fix errors, without really understanding the problem or the impact of their changes. They have an unsystematic approach whereby they start coding by trying different things without following the logical sequence of the program. They practice dependent debugging by repeatedly seeking help from others to fix coding problems. They try to code in big chunks, instead of working on a section of code at a time. They skip doing their learning tasks and begin their assignments with misunderstood or incomplete learning tasks.
Patterns of Collaboration describe the different ways students interact with, exchange information and seek assistance from others. It highlights and describes the different levels of dependency and how much students rely on their personal networks as they learn programming. Students’ personal networks include their peers or friends in the same class, in the same degree, friends from other degrees, friends in industry, members of their family, as well as teaching staff.

This study identified four distinct levels in which students interact with their peers as an information source, primarily based on the dependency between the student seeking help or information and the person providing the information or help (see Figure 5-1).

---

**Figure 5-1 - Model of Dependency**
At the base level there is One Way Dependency, where the student seeking help is highly reliant on their peer(s) as an information source. In the middle level, there is Two Way Co-Dependency, where each student is intermediately reliant on each other as an information source and at the top level there is Collaborative Independent, where each student is minimally reliant on their peer(s) as an information source but they display an active interaction with them. Separately, there are also Solitary Independents, who are also minimally or non-reliant on their peers as an information source but, unlike the Collaborative Independents, they interact minimally with their peers.

### 5.2.1 One Way Dependent

In the One Way Dependency type of interaction, the peer is the preferred provider of information and the student is the receiver of the information. The student seeking help is mostly or totally dependent on the peer providing help with a programming issue. There is a heavy reliance on the information provider, where the peer acts as a crutch or as a delegation service to the student who is seeking help. Primarily, these students seek and prefer help face to face, both from teaching staff and peers.

There is minimal participation or interaction from the student seeking help. Usually, the student presents a query and the help provider gives the student a solution to the problem, together with a brief explanation. The student may or may not fully understand the explanation and in some cases the student is only interested in getting the problem solved. Similarly, the student may or may not be fully able to replicate independently what was shown to them in the way of assistance.

A One Way Dependency interaction is likely to begin inside the tutorial class when the student is seeking an immediate guidance or a solution to a problem from a peer while the tutor is unavailable. This may involve asking the person sitting next to them or asking a peer who is perceived to be knowledgeable in the field. These irregular interactions tend to develop into more regular consultations taking place outside the classroom, where consultation may be done face to face or via online media, such as by text message, Facebook or email.

“Outside of tutes I just contacted my friends via text message, Facebook or email. At some points I was pretty much zipping up the file and sending it off to a friend so that they could open it up and be able to solve what I had done wrong. He was able to give me a quick explanation on what he had done to solve the issue. At that time I did understand however some of the times when I asked different friends, they were not always able to explain it in a way that I could understand.” (I21)
Students who are One Way Dependent tend to seek assistance from multiple information providers, iterating from one source to another for each consecutive problem they encounter. This tendency is observed especially in students who have weak problem solving skills. When a different subsequent problem is encountered, if the student does not feel comfortable asking the previous source again, a different information provider is sought, which may be a different peer or a personal contact from the student’s personal network or a different teaching staff person. For example, the student may iterate between several peers, the class tutor and the help lab tutor.

“Where can I go? Well you can only see the lecturer for a certain time and then that is it! Or you can only come to the help labs for a certain hour and then you would be off by yourself trying to figure out things.” (I4)

In some cases where the student may feel apprehension that their request may be construed as potential plagiarism, especially with assignment deliverables, they may be inclined to consult their wider personal networks such as friends or personal contacts from a different degree or university.

“I had friends in class that I contacted, but assignments are individual tasks. I can’t really ask what exactly they had done. Luckily, when I needed to find out how to work around the problem, I had another friend that helped me out who was doing his masters.”(I21)

5.2.2 Two Way Co-Dependency

In a Two Way Co-Dependency type of interaction between the student and his/her peers, both parties behave as providers and recipients of a different set of information exchanged between them, each relying or co-dependant on the other. Students may be co-dependent because each has specific skills or knowledge the other lacks (skill co-dependency) or they may consciously divide tasks between them to get them done more quickly (task co-dependency).
Skill Co-Dependency

In skill Co-dependency, students seek assistance for a particular phase in the programming cycle where they have weak skills. Debugging is the programming phase where assistance from peers is most often sought, as reported by students in this study. However, information seeking also occurs for how to get started, planning (i.e. structuring an algorithm) and especially for the most challenging parts of a deliverable.

"Sometimes, some of the people from our study group just gave me the code. When time was a huge factor, under pressure you just wanted to make it work and it didn’t really matter, I just wanted the code. There was however another member in our group who did not give you the code. He taught you ways to do it and he made sure that you understood what was going on." (I26)

Unlike the One Way Dependency interaction, the Co-dependent interaction is less spur of the moment. The Co-Dependent interaction is also more evolved. At this point, the student seeking information is fully aware that the peer from whom information or help is sought has the skill they are seeking help with.

"I probably needed assistance with debugging more than anyone. There was someone that was at guru level and we would all learn from him. We used him to explain to us instead of us doing it without knowing and we also tried to learn from that as well." (I10)

Task Co-Dependency / Division of Labour

The task oriented co-dependency (or division of labour) is more formal in nature compared to the skill based. It is based on an agreement between two or more students where each party chooses to complete or attempt to solve a problem for a different deliverable task (e.g. tutorial or assignment). Both parties then exchange results and an explanation of what was done. The recipient in each case may or may not be able to fully understand or replicate what was done by the other student. In this scenario, when the students are still programming novices, debugging and working out problems takes longer so in order to speed up and get to complete each tutorial, some students may opt for this type of division of labour. Both recipient and provider are mutually reliant on each other.
“I did have a good friend in the class and we helped each other out with working out errors and things we didn’t understand. Towards the end, we were both at a dead end with errors in the tutes so each of us worked on separate tasks so we would get all the tasks completed. Then we would explain out what we had done to get it working, but sometimes it wasn’t clear depending on the topic.” (I21)

5.2.3 Collaborative Independent

The Collaborative Independent interaction has a higher level of collaboration between the student and the peer, friend or personal contact. There is a stronger two way interaction or exchange of information. Both parties complement each other. They are both recipients and information providers to each other as they explain to each other what coding problems they are having and how they have approached or solved a problem.

“When I had a problem, after looking at my notes and several textbooks I would then talk it out with my friends. Sometimes they’re willing to give you the answer, sometimes they’re not. Sometimes when I think about it and I talk it out, I might actually find the answer myself because I’ve actually thought about it.” (I17)

In this scenario, the student has completed a task or subtask but is looking to understand it better by comparing what they have done with their friend or personal contact. Each party shows the other what they have done and they explain to each other. The student looks at their friend’s approach more critically, thoroughly comparing to what they have done. They are highly reluctant to take or adopt anything they don’t thoroughly understand. Often they choose to keep the way they have approached their task as they understand it better.

“I have a friend that I would work with side by side in class and outside class. We try and work out problems and help each other. When I explain things to her, it helps me understand it better. It shows that I actually understand it myself. Her method to me seems a bit more complicated though so I keep my own way.” (I28)
An evolved two way interaction where real collaborative learning takes place is when the students choose and trust their friends or personal contacts to exchange approaches to debugging, solving a coding issue or even structuring an algorithm. The exchange is not merely an exchange of information but each party critically compares and even tries to incorporate what was shown in order to understand how it works. Each student independently replicates what was shown to them in order to gain further insight and choose which option they critically deem the best.

“Me and my friends would work together for assignments. We would figure out how it is done and what is needed, then we would go our separate ways and create our own versions. We share our knowledge and learn from each other. Often I would talk to my friends and we would discuss bugs together. I tried implementing each different suggestion so if one didn’t work I would try the next one.” (I2)

Students who interact collaborative independently with their peers or friends, seek a pointer or direction rather than a complete solution. Their ownership of their learning is stronger as well as their problem solving skills.

“My friend who works in industry gives broader, general help. He will give you the framework of how it should be. He doesn’t work through step-by-step. I might show him some of my code and he will point out the part that doesn’t look right.” (I29)

5.2.4 Solitary Independent

Solitary Independents are very similar to collaborative independent students, in that they help and guide other students, but they themselves tend to not seek help from their peers. Highly Engaged and Independent learners are sometimes reluctant to ask their peers for fear of intimidating and confusing them or because they are working ahead and their peers are not up to the same stage they are at.

“I tried asking some of my friends but it was just a waste of time! I think maybe I was a bit too far ahead of my friends. So, my last resort would be to ask the tutor.” (I14)

They may help each other by looking at each other’s errors as just another pair of eyes but they would only consider consulting their trusted set of friends or personal networks for things like planning, structuring an algorithm or a debugging approach.
Also, students who are solitary independent learners have a strong ownership of their learning and may feel very comfortable just using Google as one of their preferred sources of information when solving programming problems instead of seeking help from their peers. They are confident that they can obtain fast results from Google because they know what they are looking for. They have a clear mental model of what they are trying to do and can successfully identify the specific parts from their search results and incorporate these into what they are trying to solve. They may ask teaching staff for help if they are in class. Some of these students feel happy to explain things to other students when help is sought from them but when they have a problem they prefer to seek online help using internal and external forums or their tutors, rather than asking their peers for help.

“I don’t know anyone in programming. Generally a search on Google would fix my problem much faster. Some problems can be quickly looked up, while other problems would take longer. For these you would have to search several sources before finding a conclusion that fits your problem. There was never a problem where I would take hours - eventually I would have figured it out.” (I22)

The independent students (collaborative independent and solitary independent) tend to be the students that help their peers and friends in their personal networks and study groups. The other students seek help from these students when they are doing their first programming course. Many of these independent students end up working as help lab tutors in their second and final year of their degree. One characteristic of this group of students is that they learn by helping others.

“I had three other friends and we would just catch to do our homework together for all subjects. We each had our own strengths. We encouraged each other to finish each of our homework that we hadn’t finished. It wasn’t mainly focused on Programming, but yeah, they always had questions and I’d always end up helping them with it.” (I14)

Seven (of 31) participants in this study were identified as solitary independent.
5.2.5 Variations

Students can interact with multiple peers or personal networks concurrently, at different levels, either in the same course they are working on or in different courses. For example, a student can work co-dependently with a peer using task oriented division of labour and at the same time they can interact in a One Way Dependent fashion with a friend from their wider personal network.

A student may interact in a collaborative independent fashion with their personal networks within their degree, for courses they perceive to have a higher personal relevance and so they are willing to take a stronger ownership. However, for courses they perceive as having a lower personal relevance, they revert back to a One Way Dependency interaction with their personal networks.

This change of interaction between student and personal networks may happen because they move from an introductory course to the next course which they find harder. It may also happen between different courses where the student sees a different Perceived Personal Relevance. For example, a student may have a low personal relevance for programming courses but a higher personal relevance with courses such as Databases or Networks.

“Some of the errors in programming are hard to spot because I would look at it and I wouldn’t know. It would take the lecturer or the tutor to come over and, point out what I’ve done wrong because I could not see it. I would also get my friends to look at it and we would compare code. With the other IT subjects (Database), sometimes my way might be better or shorter and sometimes theirs might be better so we compare, and reflect ‘Oh we could have done that to make it work.’” (125)

The tendency noted from what the students in this study have described is that the deep learners who are highly engaged seem to remain at a higher level of collaborative learning and minimal reliance on their peers whereas the intermediately and minimally engaged seem to move upwards and downwards in the way they interact and rely on their peers and personal networks. However, the students who genuinely take ownership of their learning and actively seek to understand and learn from all their sources of information may start as novices with One Way Dependent interaction. As they learn how to approach programming problems, however, they move towards a Collaborative Independent type of interaction with their peers / friends. Once they reach this point they don’t seem to revert back.
5.2.6 Summary

Patterns of Collaboration describe the different ways students interact with, exchange information and seek assistance from others. It highlights the different levels of dependency, and ultimately, how much students rely on their personal networks as they proceed to learn how to solve programming problems. Students’ personal networks include their peers or friends in the same class, in the same degree, friends from other degrees, friends in industry, members of their family, as well as teaching staff.

Students interact with all or most of their personal networks but each student profile does so in different ways and with different levels of dependency. This study identified four distinct levels of dependency in which students interact with their peers as an information source.

In a One Way Dependency, the student seeking help is mostly or totally dependent on the peer, who acts as a crutch or as delegation service. Often, the student may not fully understand the help received and be only interested in getting the problem solved. Interactions begin inside the classroom when the student seeks help from a peer and then develop into more regular consultations outside the classroom, which might be face to face or using online media. These students often seek assistance from multiple information providers, iterating from one source to another. In some cases they consult their wider personal networks where their request for help may be construed as potential plagiarism. These students predominately prefer face to face interaction.

In a Two Way Co-Dependency, students rely on or are co-dependant on each other. Students may be co-dependent because each has specific skills or knowledge the other lacks (skill co-dependency) or they may consciously divide tasks between them to get them done more quickly (task co-dependency). With Skill Co-dependency, students may seek assistance for phases in the programming cycle where they have weak skills, such as planning, coding or debugging. Task Co-Dependency (or division of labour) is more deliberate in nature, where students agree to complete different deliverable tasks and then exchange the results and an explanation of what was done.

Collaborative Independent students practice a stronger two way interaction or exchange of information. Both parties complement each other. They explain to each other what coding problems they are having and how they have approached or solved a problem. Often the students have completed a task and show each other what they have done in order to gain a better understanding. These students tend to collaborate with peers that they trust and are at a similar skill level.
Solitary Independent students are similar to collaborative independents, in that they help and guide other students, but they themselves tend to not seek help from their peers. These students have a high ownership of their learning and can independently solve problems more quickly using Google and online forums than by seeking help from others. They tend to seek help from teaching staff as a last resort. These students tend to be the help providers that other students seek help from. Many of these independent students end up working as help lab tutors in later years of their degree. They also learn by helping others.

Students can interact with multiple peers or personal networks concurrently, at different levels. They might work co-dependently with a peer in the same course and at the same time consult a friend from their wider personal network in a one way dependency. Some might be collaborative independent in courses they perceive to have a higher personal relevance in but revert to a one way dependency in courses they do not see as relevant to them.

The collaborative and solitary independent students tend to be the highly engaged students with a high Perceived Personal Relevance, whereas the lesser engaged students with lower Perceived Personal Relevance tend to move upwards and downwards in the way they interact with their peers.
5.3 Patterns of Information Use

Patterns of Information Use describes the different ways students use written and online sources as they learn how to solve programming problems. At a basic level, students look up their information sources for reference checking, including syntax verification and finding the meaning of errors. Syntax verification involves verifying correct formats for constructs, functions and components and their correct use and structure. Checking meaning of errors involves checking various online sources to understand the meaning of the error message displayed by a compiler.

Solution discovery is looking through the sources information in search of a solution to the particular problem the student is working on, particularly assignments. The type of solution the student looks for varies on their skill level and goal. Looking for Sample Answers is the simpler form of solution discovery that involves looking for a section of code that matches the students’ problem.

Code Adaptation, Comparative Analysis and Beyond the Scope are the more elaborate forms of solution discovery. Code adaptation involves identifying and understanding parts of subsections of code written by others that could be adapted to one’s own code. Comparative Analysis involves students comparing their code to others (personal networks and online forums) with the intention to improve their own coding style. Beyond the scope involves using online sources to learn new languages or advanced features that are not covered in the curriculum.

5.3.1 Syntax Verification

All students use various sources to check for correct syntax during the process of coding. Syntax verification can range from checking basic constructs, such as IF statement and loops, how to use an inbuilt function and set up its parameters to checking on how to use different inbuilt or third party components and their corresponding methods.
All the students described checking their written sources, especially tutorial notes, for syntax verification of basic programming constructs and when they had syntax errors. Online sources, however, were used differently by different programmer learner profiles to verify syntax at various stages of their study.

Typically the Beginners (Reluctant and Willing Beginners in particular) use the online sources of information as an on-line manual to help them review correct syntax formats (structure and use) for sequential, decision and iteration constructs. For example, they would verify how to write the syntax for a conditional loop and also verify if the condition is set up correctly to terminate the loop.

“I would look at Google just to see what I should be using when I was writing a loop because I didn't know how to work out the syntax to use the Do While and Do Until Loop.” (I16)

Most of the Beginner profiles (The Reluctant and Willing Beginners) also do basic syntax checking by comparing their code word for word or character by character to sample code. Some may also refer back to their written notes to check if they have missed something.

“I recall using Google as much as I could as a resource in regards to trying to work out how to fix a problem from the error messages. After I looked at Google, I would go through comparing what I had typed to what I had searched on Google to see if I could pick up any errors or typing mistakes.” (I21)

They may also verify what parameters to use in a function call and how to set them up. They use online sources to find sample code for how to implement specific functionality, which they may or may not be able to fully understand.

“I used Google to check for basic syntax to use. Simple things like which exact coding to use with the MID function. For more complicated things like creating HTML from a program, I would ask my tutor.” (I28)

The borderline competent profiles (the Keen Beginners and Budding Managers) also use online sources to verify syntax in a similar fashion. These two profiles are able, however, to better identify relevant sample code structure to verify with their own code.
The more competent and advanced skilled profiles also used online sources to verify syntax but used it as a reference to learn how to use advanced components and their related methods. These students described not needing to verify syntax for basic coding constructs online. They instead verify basic syntax by reading their written notes or following the prompts provided by the programming environment tool (Intellisense).

“To look up how to use a function like MID, what parameters and what would it return, I would use the environment Intellisense (that comes up when you start typing). I don’t think I ever really looked back at the documentation from MSDN for these things.” (I8)

“I would do random Google entries to find specific errors. I wouldn’t ask ‘How do you write a loop?’, for those sort of things, I would try to understand myself.” (I8)

### 5.3.2 Finding the Meaning of Error Messages

Seeking understanding for the meaning of an error message displayed by a compiler is the first and most common step students described doing, particularly in the early stages of learning how to program. All students described using the Google search engine and typing in the error message word for word, in order to find out the meaning of it. Different Programming Learner Profiles, however, dealt with, interpreted and acted on the information found online in different ways.

Most of The Reluctant and Willing Beginners, who initially searched their compiler generated error message on Google, described reaching some understanding as to the meaning of the error message. However, they described being unable to then find what was triggering the error in their code and requiring guidance from teaching staff or personal networks to make sense of how the information that was searched could be used to identify the problem code that was triggering the problem. These students lack the familiarity and debugging practice as well as a good mental picture of the task at hand in order to relate the searched information to their problem code. Nine students (out of 31) described not being able to act on the error message that was searched on Google.
“When I was getting an error and not understanding why that error was there, I used Google looking for an answer but I found many highly advanced answers that I did not understand. Even the Microsoft help wasn’t helpful because I could not understand it.” (I20)

“I would Google some Java code commands to understand what the error was but it was difficult to get a clear picture of what caused the error. The issue being that the error messages are so generalised and many things can cause that type of error.” (I9)

Those students bordering competency, mostly Keen Beginners and some Budding Managers, described being able to understand and fix their coding sometimes. However other times, for more complex problems, they described not being able to follow instructions posted on forums (7 out of 31).

“In the Stack Overflow forum, there were many posts that would show the code for a problem and other people’s feedback or comments on how to fix it. I thought I could figure out my problem but I followed the instructions and it didn’t work for me.” (I26)

The more competent skilled and advanced profiles described initially searching the error message on Google and then from there, finding forums that they would later go to directly. They described being able to make sense of the information that was searched and also be able to not only follow instructions found on forums on how to fix a similar problem but actually being able to fix their problem (12 out of 31). Some of these students also described reading how others had fixed a similar problem and comparing to how they had fixed it in an attempt to find better ways of fixing problems (5 out of 31).

“A lot of time when my process was failing and I had an error message, my first instinct would be to Google. Most times I would find an example someone else had problems, and from that I could see how to fix mine.” (I8)

“I don’t think I’ve ever really had a coding issue that was just a complete dead end. If there was a massive problem, looking it up on the Internet seemed to work. Somebody would have already asked that question for a similar kind of problem on a forum and they would have also posted how they resolved it.” (I22)
5.3.3 Looking For Sample Answers

Looking for sample answers is the simpler form of solution discovery that involves students searching for a section of code that matches the problem they are trying to solve, particularly for assignments. The type of answer they look for depends on their skill level and goal. While novices tend to look for a complete answer (i.e. section of code), more competent and advanced students look for smaller parts or small segments of code - pointers or directions to help with their problems.

Most novice programming students generally search for information to help them find a solution to a task, mainly assignments. They encounter difficulties when both looking at their written notes and also online. When referring to their written notes, they are unable to recognise the parts that they can use or adapt, often glancing through them superficially.

“Saving to a text file and using the arrays – felt really unexplored and I had to Google a lot to understand it. I went through a lot of the lectures for it but I couldn’t really find it.” (I20)

When seeking complete answers online, they also find it difficult to understand the information they find and how they can adapt it to solve their problem.

“I need someone to explain it to me. There’s so much information on the Internet and still even though I’m getting it off the Internet, I may not understand it.” (I25)

When searching online, there is a lot of information that the student needs to sift through in order to identify what may be relevant. Their keyword search needs to be specific to limit the excessive hits returned by a Google search. As there are many programming languages, the student would need to specify the language they are working with to narrow down their search results.

“I have had to Google some of the code but the results did not really come up with the best answer, especially with coding. You might get mySQL coding examples instead of VB coding examples.” (I5)

They also require learning programming lingo to be able to type up their search questions and also interpret the results accordingly as well as knowing which forums would be good to look at for their knowledge of programming.
“The Internet had everything. It allowed me to look at different things and how they were done but the challenging thing initially was learning the lingo. That sort of took a while. When I could work out my specific question, I could find my answer a lot easier.” (I15)

Students may also have limited time, do not know how to solve the problem or write an algorithm. They may also be disengaged with the task. These reasons will prompt novice students even further to look for a complete answer.

“Staying up late, looking on Google using it as my research tool, trying to access knowledge via the Internet that may or may not be there is a stressful way of completing an assignment I found. There is so much stuff out there particularly in Java.” (I9)

Identifying what is relevant is in itself a challenge for a novice programming student as they find it difficult to identify the parts that may be applicable to their problem. When they refer back to their written notes (i.e. tutorial notes) they would have had to have completed it but also have understood it in order to recognise the code that would be applicable to use in their assignment.

When they search on Google for sample code it is unlikely that they will find an exact fit to their problem. In many cases, the sample code that is found through forums may be incomplete, such as missing the variable declarations. It may also require some adaptation to fit the students’ problem.

They may also have an unclear mental picture of the programming task at hand they need to do, making the identification of sample code they find and understanding of it even more difficult. This renders them even more unable to recognise specific code parts that are useful to their problem from the found sample code.

“Sometimes I didn’t know what I would be looking for because I didn’t understand what I was being asked to do.” (I9)

“I don’t feel confident that I can process the information in a way that makes sense to what I am looking to do, and again it comes back to being a foreign language.” (I9)

The sample code and advice they may find on forums may also be too advanced for a novice programming student to be able to identify the parts that may be relevant to their problem.

“I Googled, but a lot of it, I didn’t understand the concept of what they were trying to explain because they were on different levels of what I was.” (I26)
The intermediately competent profiles such as the Budding Managers and some Budding Practitioners, described looking for sample code with parts of the answer, particularly for the more complex tasks. Some prefer looking for sample code online as an alternative to adapting code from their tutorials and notes as they believe it is faster. This may be because they skipped the planning, they may be under time pressure or they cannot conceptually identify how to adapt code from their written notes or tutorials for their assignment (9 out of 31).

“Under a rush you just go straight for it. Asking your friend or anyone you know, using as much resources as you can to find out what’s going on. Google especially. Google has sample code whereas the course notes you had to kind of read and work it out, so you had to read everything. Google was the easiest way out.” (I26)

Students from the more competent and advanced profiles (Keen Beginners, Budding Practitioners, Budding and Advanced Developers) also search for coding examples both from their written notes and online, but they are searching for very specific pointers and not complete answers. They are seeking a second opinion or better ways to approach a coding problem compared to their current approach (10 out of 31).

“If I don’t understand or I haven’t tried something hands-on, I would look it up on the web. First, I would check the lecture notes and the lecture notes are good, but I like then having another description from another person. Both of these provide different examples that would help me understand what I’d be using. Our University is not the only university teaching Accounting or IT etc.” (I13)

These students also described searching for coding examples in order to gain a preliminary understanding in regards to a task or a coding issue or because they were working ahead of the class. They also seek to work out a problem as best they can prior to asking teaching staff.

“I would Google first then I would ask on the internal forum because I felt the Lecturer was a last resort, I wanted to learn stuff by myself.” (I26)
5.3.4 Code Adaptation

Code adaptation is more evolved form of solution discovery and involves identifying parts or subsections of code, written by others, that could potentially be adapted for use in their own code. Once the code or relevant sections have been identified, the student needs to be able to extract those code lines and adjust them to fit their program. Making the adjustment is the most challenging aspect, as the student needs to be able to understand not only their own code but also code written by other people and how to modify it. Students generally adapt code from their written sources (e.g. tutorial exercises) and also from online sources.

Whilst adapting from online sources may initially be appealing to the beginner profiles, as there is vast amounts of code available, in reality they find it is the most challenging. Many Reluctant and some Willing Beginners tend to jump straight into a Google search to find some sample code but then find integrating it into their own code difficult.

“Sometimes, I still don’t understand the examples on Google. They sometimes give you an example of how to use syntax or some function, how to write it, but I wouldn’t understand some of the code. Because I have not learnt it before, I wouldn’t know how to change that to something that I understood.” (I28)

Some of these students struggle with carrying out a simple form of code adaptation if the sample code does not include the variable declaration section at the beginning of coding section. They are unable to recognise what variables they need to declare and in some cases what variable type they should use. This can lead to multiple logical and syntactical errors that a beginner student can find quite frustrating.

“At first it is like, ‘Ah, that’s how they did it,’ but then you realise that whatever you copied or analysed from online doesn’t work on yours. That part of the code you used could have a previous part to it like the declared variables at the top, and if I did not know what variables they declared at top, the code would never work so that became quite frustrating.” (I27)

The Keen Beginners described being able to adapt code from online sources but found the process time consuming because initially in their searching they did not narrow down language specific examples.
“When I used Google to look for sample code, I typed VB coding but the coding examples that came up had a lot of variations and some of them were for a different programming language. It took me a lot of time trying to find out how one code works and this is probably the frustrating part.” (I5)

Other intermediately competent profiles such as the Budding Manager and Budding Practitioners described being able to adapt code for simpler functionality but found adapting code for more complex coding problems, such as working with abstracted functions, more difficult.

“It was quite intensive going through forums asking people how things work and seeing some examples of work from other people. Some were helpful, but others were not because it might’ve been stuff that I’ve already tried and it didn’t work.” (I2)

Most of the students who described a positive experience adapting code were from the Budding Practitioner, Budding Developer and Advanced Developer profiles. These students described, as a starting point, revising their written notes and, in particular, their completed tutorial work looking for similarities to their problem. They also described looking at postings on external forums and being able to identify the relevant pieces of sample code solutions and being able to successfully adapt it to their problem.

“Well I would start by looking at my [completed] weekly tutorial work. I would run them and look for ones similar to my problem. I would look at the tutorial more closely and compare it. If I cannot find similar tutorials to my problem, then I would look up on Google.” (I12)

5.3.5 Comparative Analysis

Comparative analysis is a more elaborate form of solution discovery that involves students comparing their code to others, to see how others have implemented or solved a similar problem, with the intention of learning from and improving their own coding style. In this process they also find and learn better techniques and advanced features that they could incorporate into their course deliverables. This type of activity is mainly practiced by the more competent and advanced profiles such as Keen Beginners, Budding Practitioners, Budding Developers and Advanced Developers (10 out of 26).
The students compare their code to sample code from various sources looking for a different perspective. They may start by looking at their completed tutorials and lecture notes. Only a few students discussed referring to their textbook.

“I would look at my notes and then I would look at the textbooks, then I might have gone to the library to see if there’s another programming book that will probably give me a different perspective.” (I17)

They also compare their code to sample code found on the Internet. When looking at online sources to do comparative analysis, they primarily would look at postings from external forums but they would also look at explanations or tutorials of what other people do or how others solve a similar problem.

“If I was at home, I would search Stack Overflow. If the task was about arrays, then I would try to find background information, a general idea to make sure that I am doing things correctly but not a specific solution. I would then pull a bit of information from each problem which helped with understanding what needed to be done.” (I29)

Most of these students also compare their code to code written by other students. In particular, they discuss and compare code with students in their personal networks - those students they believe understand the issues they themselves are interested in.

5.3.6 Beyond the Scope

The more highly competent profiles (Budding and Advanced Developers) described going beyond the scope of their studies to enhance their understanding. They seek additional or complimentary information to their written notes by looking at various online sources. These students described using external forums to learn new languages or advanced features (5 out of 31).

“I would search based on the feature I want to learn, for example, how to use a chart in VB.Net. In some cases they give you a short tutorial.” (I12)

“For later phases in the assignment that were more complicated, I always looked at Google because it was a learning curve for me. Also, even if I didn’t have an issue, I was a bit curious to see what other things are out there because I was just interested and I wanted to improve what I was doing” (I15)
5.3.7 Summary

Patterns of Information Use describe the different ways students use written and online sources as they proceed to learn how to solve programming problems. All students use all sources of information, written and online, but use them differently.

At a basic level, students look up their information sources for reference checking. This typically includes syntax verification and finding the meaning of errors. Syntax verification involves verifying correct formats for constructs, functions and components and their correct use and structure, from both written and online sources. Reluctant and Willing Beginners use online sources as an online manual to verify basic syntax. Keen Beginners and Budding Managers do the same, but can more readily identify relevant sample code structures that they can use. The more competent and advanced profiles, in contrast, use their programming tool to verify syntax while coding and only use online sources for checking on advanced features and components.

Finding the meaning of errors involves checking various online sources to understand the meaning of the error message displayed by a compiler. All students use Google to do this, but the beginners tend to have difficulty identifying what is triggering the error in their code and how to fix it. Keen Beginners and Budding Managers can understand what they find online and are able to fix simple coding problems, but find it difficult to fix more complex problems. The more competent and advanced profiles become proficient at using online forums to search for errors and fixes, as well as using Google.

Information sources are also used for solution discovery, which is searching for a solution to the particular problem the student is working on, particularly assignments. The type of solution the student looks for varies on their skill level and goal.

Looking for Sample Answers is the simpler form of solution discovery that involves looking for a section of code that matches the students’ problem. Novices tend to look for complete answers, mostly online, but have difficulty understanding and using what they find. They lack the searching skills to narrow down their searches and also lack the language domain to identify the sample code relevant to their problem and how to incorporate it into their own code. In contrast, the more competent and advanced students search for smaller parts, pointers or directions and are better able to identify and use the code they find.
Code adaptation is an evolved solution discovery that involves identifying and understanding parts of subsections of code written by others that could be adapted to one’s own code. Code adjustments are usually done from both tutorial exercises as well as from online sources. Reluctant and Willing Beginners find adapting even the simplest code difficult even though they tend to look up Google in search of sample code. The Keen Beginners are able to do it for simple tasks but find the process time consuming whilst the Budding Managers find adapting more complex coding problems such as abstracted functions more difficult. Budding Practitioners, Budding Developers and Advanced Developers described a positive experience adapting code from their written notes, tutorial exercises and from external forums.

Comparative Analysis is a more elaborate form of solution discovery that involves students comparing their code to others (personal networks, online forums and written materials) with the intention of improving their own coding style. In this process they find and learn better techniques and advanced features. This is mainly practiced by Keen Beginners and Budding Practitioners but it is more prominent in the Budding and Advanced Developers. When they compare and discuss issues with their personal networks they specifically do so with those they trust.

Beyond the scope is a further evolved solution discovery that involves using online sources to learn new languages or advanced features that are not covered in the curriculum. These solution discoveries are mainly carried out by the more competent and advanced profiles.
5.4 Chapter Summary

This chapter continued the in depth discussion of the Model of Influences on Learning Programming. It focussed on the Learning Behaviours component of the Programming Learner Profiles.

The concepts within Core Learning Perspective were explored: Ownership of Learning, Learning Task Intent and Problem Solving Behaviours. The chapter also explored Patterns of Collaboration, which included the Model of Dependency that shows the levels of dependency inherent in the way students interact with their Personal Networks. Lastly this chapter explored Patterns of Information Use, which described the different ways in which students use Information Sources.

Chapter 6 continues the elaboration of the Model of Influences on Learning Programming. It explores the Programming Learner Profiles in depth, showing that students are quite different, depending on their Perceived Personal Relevance, Learning Trait and Skill Level. It presents the seven learner profiles that were developed during the analysis. It then explores the relationships between all of the concepts in the model in depth.
Chapter 6  Programming Learner Profiles

Chapter 5 continued the discussion of the Model of Influences on Learning Programming, focusing on the Learning Behaviours component of the Programming Learner Profiles. It explored the concepts within Core Learning Perspective, which include Ownership of Learning, Learning Task Intent and the many Problem Solving Behaviours students practise. Then the chapter explored Patterns of Collaboration, which included the Model of Dependency that was developed to show the levels of dependency inherent in the way different students interact with their Personal Networks. Lastly chapter 5 explored Patterns of Information Use, which describes the ways in which students use the various Information Sources available to them.

This chapter describes the core of the model, Programming Learner Profiles, and also presents a discussion of the relationships between the various elements of the model. Programming Learner Profiles describe the varying patterns of behaviour in the way students approach the learning of programming and the way they use their information sources and personal networks. These Programming Learner Profiles evolved during the analysis and help describe the many interrelated influences on the student learning experience and their patterns of behaviour in a unified and cohesive way. Section 6.1 presents a more detailed description of how Programming Learner Profiles evolved and developed during the analysis process.

The Programming Learner Profiles are made up of two core parts. Firstly is the Learner Nature, which encapsulates the three key defining concepts of Perceived Personal Relevance, Learning Trait and Skill Level. These are further elaborated in section 6.1. Combinations of the values from each of these three defining concepts were used to define seven distinct profiles: Reluctant Beginner, Willing Beginner, Keen Beginner, Budding Manager, Budding Practitioner, Budding Developer and Advanced Developer. Each profile describes, in essence, the nature of a different type of student. For example, students who see programming as relevant to their future, learn independently and are novices, are described as Keen Beginners. Section 6.2 gives an in depth description of each of the seven profiles.

The second major part of the learner profiles is the Student Learning Behaviour, which includes Core Learning Perspective, Patterns of Collaboration and Patterns of Information Use. There were elaborated in depth in chapter 5. As each profile is described in section 6.2, the different learning behaviours of each profile are also described, giving an overview of what each student profile does: the problem solving behaviours they carry out, how they interact with their personal networks and how they use their information sources.
During the analysis, some trends were identified from descriptions students gave of their past and present behaviours and experiences. It became apparent that there were some pathways that students tend to follow in relation to moving from one profile to another. Section 6.3 gives a brief description of the more prominent pathway progressions that were discussed by the students.

The Programming Learner Profiles concept, described in sections 6.1 and 6.2, inherently encapsulate the many relationships between the concepts within Learner Nature and Learning Behaviours. In addition to these, section 6.4 elaborates on specific relationships found between concepts in the model. These include how the concepts within Learner Nature influence Learning Behaviours (particularly Perceived Personal Relevance) and how the development of Learning Behaviours, in turn, influences the Learner Nature. Relationships within Learning Behaviours are also described, including interrelationships within the Core Learning Perspectives concept and how Core Learning Perspectives influences Patterns of Collaboration and Patterns of Information Use. The last part of section 6.4 presents two tables that summarise each learner profile’s preferences for Patterns of Collaboration and Patterns of Information Use.
6.1 Programming Learner Profile

During the analysis, three distinct dimensions surfaced as being key influences in the way students approach their learning of programming, the information sources they use and their learning behaviours. These three key Learner Nature dimensions are:

- Perceived Personal Relevance (High / Low)
- Learning Trait (independent / dependent)
- Skill Level in Programming (novice / competent / advanced)

Through the analysis process, it was found that the above three dimensions, in various combinations, have a profound influence on the student’s patterns of learning behaviour. Although all the students reported using the same sources of information and personal networks (e.g. written notes, online resources, peers and teaching staff), the way they used them varied widely, depending on their Perceived Personal Relevance, Learning Trait and Skill Level at the time. As the theory developed, this group of concepts was given the name of Learner Nature. Relationships were also observed between these three dimensions and the various approaches students take to programming tasks and the way they go about problem solving.

Considering different combinations of these three key dimensions led to the development of seven distinct Programming Learner Profiles which best describe these varying patterns of behaviour. Within each profile, the following concepts were used to contrast the similarities and differences in the patterns of behaviour between the profiles:

- Core Learning Perspectives (Ownership of Learning, Learning Task Intent and Problem Solving Behaviours)
- Sources of Information used and Patterns of Information Use
- Peers and Personal networks used and Patterns of Collaboration
The Programming Learner Profiles encapsulate the relationships and influences between the three Learner Nature dimensions (Perceived Personal Relevance, Learning Trait and Skill Level) and the patterns of learning behaviour (Core Learning Perspective, Patterns of Collaboration and Patterns of Information Use). Each profile has also been given a name, which helps to further understand these relationships.

These Programming Learner Profiles emerged over a number of iterations during the analysis process. In the interviews, students were asked to describe how they went about solving programming problems, what their experience was like and how they felt about it.

The initial stage of the Grounded Theory analysis generated a large number of concepts from the interviews. One of these concepts was how dependent students were on others for their learning, which developed into the Dependency Model described in section 5.2. This model describes different levels of dependency and how reliant students are on their peers and personal networks: One Way Dependent, Two Way Co-Dependent, Collaborative Independent and Solitary Independent.

As part of the Grounded Theory methodology involves constant comparison of concepts and cases, a number of queries were used to compare and contrast concepts against each other, across all cases. These queries highlighted an interesting relationship between the level of dependency and the use of and preference of information sources. Each group, according to their level of dependency, were using information sources in very different ways. A deeper analysis of the quotes within this relationship also revealed that each group were using their personal networks in different ways.

In further analysis looking at similarities and differences in the way each of these three groups used their information sources, it also became apparent that those students that were dependent tended to be novices and that those who tended to be independent were more skilled. It also became apparent that the students who were most dependent also were least interested in learning programming. Therefore the realisation was made that dependency had to be looked at in conjunction with Skill Level and also Perceived Personal Relevance.
These three key dimensions became the basis of the learner profiles. In order to pinpoint the differences between different combinations of these three dimensions, a table was drawn up where further concepts were used in order to compare similarities and differences across the profiles. Every combination of the three key dimensions was mapped out. Each interview was then categorised within each of these profiles, but there were some profiles that had no interviews. There were no students with a high Perceived Personal Relevance that were also highly dependent and all of the students at an advanced Skill Level had a high Perceived Personal Relevance and were also highly independent learners.

The profiles were then narrowed down to the seven presented in this chapter. A descriptive name which highlights the key characteristics for each profile was then allocated (see Table 6-1). These seven Programming Learner Profiles synthesise and encapsulate a number of influences on the student learning experience and present them in a structured and cohesive manner.

<table>
<thead>
<tr>
<th>Profile</th>
<th>Skill Level</th>
<th>Learning Trait</th>
<th>Perceived Personal Relevance</th>
<th>Participant Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reluctant Beginner</td>
<td>Novice</td>
<td>Dependent</td>
<td>Low</td>
<td>5</td>
</tr>
<tr>
<td>Willing Beginner</td>
<td>Novice</td>
<td>Independent</td>
<td>Low</td>
<td>6</td>
</tr>
<tr>
<td>Keen Beginner</td>
<td>Novice</td>
<td>Independent</td>
<td>High</td>
<td>5</td>
</tr>
<tr>
<td>Budding Managers</td>
<td>Competent</td>
<td>Dependent</td>
<td>Low</td>
<td>4</td>
</tr>
<tr>
<td>Budding Practitioner</td>
<td>Competent</td>
<td>Independent</td>
<td>Medium</td>
<td>3</td>
</tr>
<tr>
<td>Budding Developers</td>
<td>Competent</td>
<td>Independent</td>
<td>High</td>
<td>4</td>
</tr>
<tr>
<td>Advanced Developers</td>
<td>Advanced</td>
<td>Independent</td>
<td>High</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 6-1 - Programming Learner Profiles Summary
6.1.1 Learner Nature: Perceived Personal Relevance

Perceived Personal Relevance was a key concept that emerged from the analysis of the data as having the largest influence on how much a student was interested in learning programming. The concept describes how relevant the student sees programming skills to them in the future. In the majority of cases this related to the career goals the students already had in mind while they were learning programming. A minority of students also mentioned programming being a personal interest or hobby even though they didn’t intend to make it their career.

“I really enjoyed Programming. I looked at it as like I want to know how operating systems are built, and I want to know how to build applications and even games. I’m fascinated by how you make those things. I just want to learn about Programming, and well, my passion for it only got bigger.” (I8)

“Probably I wanted to do an IT course because I consider IT to be a really growing industry and it’ll be really helpful to have an IT background, going for a job.” (I20)

“I didn’t know how programming was going to fit into a longer-term career. I knew I didn’t want to be a developer.” (I10)

The concepts of career goals and interest arose from the data during the open coding analysis. These values were then abstracted to the Perceived Personal Relevance Category. Later, once the model had been finalised, these concepts were compared and contrasted to the literature.

Career Goals

A key theme that arose from the interviews was how relevant programming skills were going to be to the student’s future career and job prospects. Students who were most interested in programming saw it as relevant to their future career goals and knew they wanted to work as programmers (12 of 31). These tended to also be the students with the highest skill levels and independent learning traits. These students were intrinsically motivated to learn programming and expressed a high level of interest and enjoyment when programming.
The students that perceived that they would never work as programmers nor do any programming in their intended careers tended to express a very low level of interest in learning programming as they did not see it as relevant to their future at all (13 of 31). These students’ career goals tended to include:

- Attaining an IT / IS degree for a better job or career prospects in management or business analysis
- Working in the IT field as IT practitioners working with technologies but not necessarily working in programming

These students were extrinsically motivated to learn programming in that their learning intent was to get the task done to achieve the marks necessary to achieve their primary career goals. They were not interested in learning programming for its inherent characteristics.

Some students had a mixed view of the relevance of programming (6 of 31). Those who wanted to be business analysts saw value in learning programming as both a pathway to their career goal and also as a necessary skill to make them better analysts. Others who wanted to work in fields such as website development or networking saw programming as a skill they needed as part of their intended career but not the main focus of their career.

**Interest**

A few students (2 of 31) expressed a personal interest in programming, even if not for a career. They said they enjoyed it and had done it before on their own and outside their studies. One of these students was interested in developing a software system for their family business, while the other was interested in game development as a hobby.

“As a hobby, I am interested in game development. I am trying to learn C# because Microsoft has this program that I can use to develop games and I could sell it.” (I2)
6.1.2 Learner Nature: Learning Trait

The Learning Trait dimension describes how dependent or reliant the students are on others (their sources of help) when working out a programming problem that may be with planning, coding or debugging. Their sources of help are their teachers (help lab tutors and class tutors) as well as their peers and their personal networks. Learning trait has two values: dependent or independent. These values were derived by summarising into two groups the four values from the Model of Dependency on Peers and Personal networks, but also including teaching staff (see section 5.2 for a detailed description of the Dependency Model).

A dependent Learning Trait describes a passive approach to completing a task, heavily relying on others. For example, moving from one source of help to another iteratively every time a problem is encountered, or the constant going back to the preferred source of help as an on-going way of completing the task.

An Independent Learning Trait describes figuring out problems mostly or totally independently. This may still include using all of the information sources (written, online, peers and teachers) but seeking only a pointer or direction or a small piece of the missing puzzle, showing pro-activeness with the help received. The students who practice independent learning in programming are also confident in carrying out other forms of independent learning such as learning from online tutorials and learning topics that are beyond the scope of their formal studies.

The values of Learning Trait (dependent and independent) were found in the data during the open coding analysis and abstracted into the concepts of Learning Trait and the Model of Dependency. Once the theory was complete, these concepts were compared and contrasted to the literature.

6.1.3 Learner Nature: Skill Level

Student programming skill levels can be described as ranging from a novice skill level that can progress towards a competent level and this can progress towards an advanced skill level. The values of novice and advanced skill levels came from the literature, while the definitions of each skill level arose from the data. Competent was added to provide an intermediate skill level.
Novice Skill Level

A novice learner in programming is someone who has just began to learn programming for the first time. They have a rudimentary understanding of basic programming constructs (sequence, decisions and repetition). They are comfortable using simple programming constructs such as if statements and basic loops (For loops) as well as using simple variables. They struggle to combine different constructs together to build an algorithm. They struggle with more complex constructs such as conditional loops and complex data structures such as arrays. They have a weak mental model of the inner workings of their program and how the algorithm is meant to execute. They can check for typos and syntax problems but find it difficult to check for logical errors (semantic problems) in their code. They are looking for the whole answer when they look for sample code. They find it difficult to adapt bits of sample code to their problem and they often cannot recognise the patterns laid out in completed tutorial work and course materials when they need to work on their assignment deliverables.

Competent Skill Level

A competent learner in programming is someone who has mastered all the programming constructs of sequence, decision and iteration. They are also comfortable using simple and complex data structures. They can confidently combine the programming constructs to build a solution algorithm for a problem. They can use inbuilt functions as well as write their own functions. They begin working at higher levels of code abstraction and writing more generic and reusable functions. They do simple planning and also have a stronger mental model of their program. They can use debugging techniques/tools (Message boxes, Print Statements, Debugger Tool and code isolation) independently to trace the execution of a program and narrow down a coding problem. They can confidently adapt bits of sample code to their problem and they can recognise the patterns laid out in completed tutorial work, course materials and Googled sample code when they need to work on their assignment deliverables.
Advanced Skill Level

An advanced learner in programming goes beyond competency level in that they can do the same but faster and more efficiently. Having surpassed competency level, another key difference is that they are more focused on how the code is structured, modularity, code reuse and architecture. Their planning of the actual coding is more detailed and their debugging steps are more thought out. At the advanced skill level, they are more likely to using desk checking to work out problems. They are quite comfortable working at the higher levels of code abstraction and multi-tiered software architectures. They are well practiced at creating programs. They go above and beyond what is required in their courses, often learning advanced features and new languages independently.

6.1.4 Learning Behaviours Overview

Within the profiles, a number of concepts were used to contrast the similarities and differences in the patterns of behaviour between the profiles. These concepts were grouped under the heading of Learning Behaviours and are described, in depth, in chapter 5.

The Core Learning Perspective concept describes the attitudes, intentions and behaviours each distinct profile has towards learning programming. This concept is composed of three sub concepts: Ownership of Learning, Learning Task Intent and Problem Solving Behaviours. The Ownership of Learning concept describes how much responsibility they take for their learning.

The Learning Task Intent concept describes the goal or intention students have when doing their learning and deliverable tasks (assignments) – whether they have a deep desire to understand the code or just simply want to get the task done. The Problem Solving Behaviours concept describes the various steps and actions students undertake when solving programming problems during their learning, which range from ad-hoc, intermediate to thorough.

The concept of Patterns of Information Use is used to compare and contrast which sources of information are preferred by each of the profiles and how they are used. Patterns of Collaboration are also used to contrast the different levels of dependency each profile has on their personal networks and how they interact with them.
6.1.5 Summary

Three distinct dimensions surfaced during the analysis as being key influences and were grouped as Learner Nature: Perceived Personal Relevance, Learning Trait and Skill Level. Relationships were also identified between these three dimensions and the various approaches students take to programming tasks and the way they go about problem solving. Considering different combinations of these three key dimensions led to the development of seven distinct Programming Learner Profiles which best describe these varying patterns of behaviour. The Programming Learner Profiles encapsulate the relationships and influences between the three Learner Nature dimensions and the patterns of learning behaviour (Core Learning Perspective, Patterns of Collaboration and Patterns of Information Use). Each profile has also been given a name, which helps to further understand these relationships.

Within Learner Nature, Perceived Personal Relevance was a key concept that emerged as having the largest influence on how much a student was interested in learning programming. Students who were most interested in programming saw it as relevant to their future career goals and knew they wanted to work as programmers. These tended to also be the students with the highest skill levels and independent learning traits. On the other hand, students that perceived that they would never work as programmers, or do any programming in their career, tended to express a very low level of interest in learning programming.

The Learning Trait dimension describes how dependent or reliant the students are on others for their learning. A dependent Learning Trait describes a passive approach to completing a task, heavily relying on others, while an Independent Learning Trait describes students figuring out problems mostly or totally independently.

Skill Level was the third dimension that made up Learner Nature and was used to define the profiles. A Novice learner in programming is someone who has just began to learn programming for the first time and have a rudimentary understanding of basic programming constructs. A Competent learner in programming is someone who has mastered all the programming constructs, simple and complex data structures and can confidently build a solution algorithm for a problem. An Advanced learner can do the same but faster and more efficiently. They are also more focused on how the code is structured, modularity, code reuse and architecture.
6.2 Seven Learner Profiles

This section presents the seven distinct Programming Learner Profiles that were developed during the analysis: Reluctant Beginner, Willing Beginner, Keen Beginner, Budding Manager, Budding Practitioner, Budding Developer and Advanced Developer. Each profile describes, in essence, the nature and behaviour of different types of students.

6.2.1 The Reluctant Beginner

The Reluctant Beginners are those students who do not see programming as being relevant, at all, to their future career goals. These students go through the learning of programming as something that needs to be done but have no real interest in learning it. Some may have chosen to do an IS/IT related degree because they are interested in learning about the field, but not necessarily about programming. There are also some that are not particularly interested in their degree but have chosen it for other reasons such as better job prospects or peer/family influence.

“I saw the growth in E-Commerce and I really wanted to be a part of it. I also wanted to stay within that business sector of my studies, because I do have a family that’s very entrepreneurial. To be brutally honest with you, I thought programming was quite boring. If you didn’t understand the basics of it, it was quite hard.” (I27)

“The IT part would never have been my first choice. I chose it because it was very high in demand, especially in my country [Asia]. IT is a must in order to get the maximum advantage from the education and also to find a job.” (I1)
They have a novice level of understanding of programming. Not all of the Reluctant Beginners are first year students, however. Some manage to get through their degree without moving beyond the Reluctant Beginner stage, learning just enough to get through each programming course they encounter. Some even manage to achieve high grades but admit they do not really understand what they learnt or cannot do programming independently.

“I got a distinction. If you ask me about the theory I can tell you about it but then I am still not confident enough in the practical work, even at this stage. Programming is super powerful. I believe not everyone is talented to do it well. I wish I was a kind of person that can do programming but I am not.” (I1)

These students have a weak learning task intent. Passing the exam or achieving good marks is their main goal. They focus on completing the task deliverables that are required rather than understanding the concepts.

“No offence, but it was one of those things that I just wanted to chalk off. I have no desire to follow it. I have no interest in it. It was all about I just got to do it to get passed and therefore I saw it as a pain in the backside.” (I9)

Reluctant Beginners are highly dependent on others for their learning (help lab tutors, class tutors, peers and personal networks both in class and outside class). They mostly prefer a face-to-face interaction with their helpers as they find it difficult to make sense of written descriptions or illustrations from their notes or even from forums (external and internal).

“I had the tutorial notes with me as well. I would go through them and look at them first, then maybe try online if the tutor is busy. But I prefer face-to-face learning. There’s so much information on the Internet and still even though I’m getting it off the Internet, I may not understand it.” (I25)

While some tend to rely on help lab tutors, others rely more on their peers and personal networks. This dependency may vary slightly over time, for example changing from preferring help lab tutors to preferring their peers and personal networks.

“I spoke to fellow class mates to ask them and they were really good and helped out on explaining what I had to do. The way they explained it made sense but it was hard. I still did not have the full concept of it because they explained it in short terms, and they helped me type it out.” (I21)
They display a weak ownership of their learning of programming. They take short cuts to get the problems solved as quickly as possible. They display ad-hoc Problem Solving Behaviours. They skip the planning and have an unsystematic approach to coding. They tend to use random trial-and-error followed up by information seeking from their sources of help. They practice dependent debugging. When they run into coding problems they spend a short time trying to work them out themselves before they seek help from others.

“With debugging, I would probably spend five to ten minutes, and if I wasn’t actually able to progress I would Google and so forth, then I would just give up, and go to the next short cut that was available, or go to university the next day and ask for help.” (I27)

They marginally use their written sources, often quick skimming and unable to identify the relevant bits that could be useful for their learning tasks. They also described disliking having to read “pages and pages” to do a task. Most of these students end up with conceptual gaps in their understanding, either because they get stuck and are unable to progress on their own, or they have not understood the help they have received. In this scenario, these students are unable to use their learning tasks as a source of information or starting point for their assignment deliverable.

“In the lectures I was taking notes, but not really sinking in. I could do some of the tutorial but not all of it would be complete by the end. It felt like a constant struggle of catching up, bridging that gap. When it came to assignments, that’s where I really struggled and I needed help from people within the class to pass those.” (I10)

They marginally use online sources of information including Google searches and posts from external forums. When using external forums, they described comparing word for word looking out for typos and looking out for what is different. They find posts from external forums difficult to understand or difficult to follow the prescriptive instructions. The same would apply with any help sought through the internal course forum. With looking up debugging errors online, they come across the meaning of the error but what they find difficult is making sense of what to look for in their actual code to fix what triggered the error.

“I recall using Google as much as I could in regards to putting in the error messages and trying to work out how to fix the problem. I would go through and look at my code then compare it to what I searched on Google to see if I’ve made a spelling mistake.” (I21)
Their understanding of how to use the debugging tool is marginal. These students tend to have a weak mental model of how the program should execute. Their high trial and error approach compounds their problem even further, making debugging even harder than what it should be and lessening their desire to learn it.

“I didn't really try using the debugger because I struggled understanding what it was saying most of the time.” (I21)

Five (of 31) participants in this study were identified as Reluctant Beginners.

6.2.2 The Willing Beginner

The Willing Beginners are those students who are willing and open to learn programming to some extent. Their career goals at this stage are wider ranging – they know they want to do something related with IT but may not be keen to become programmers as such. Some may still not be clear which IT path they will take. Their Perceived Personal Relevance for learning programming tends to be low at this stage. Some may see that programming might be relevant to their future career goals but even those that don’t are still willing to learn it.

Their interest level in programming varies widely. For some students, their interest level increases as they learn. Some may find core programming difficult but are open to mild forms of scripting and SQL. For others, as their learning progresses and the programming concepts become more complex, they may lose interest in it.

“I always got distinctions so I did well. I was fine with the programming side but the thing that put me off it was I found it really hard to get excited about it. When you do something in multimedia, you can create it and show it off to your friends. You can’t really show off an IF statement to your friends and get them excited about that.” (I11)

They have a novice skill level of programming slightly more developed than the Reluctant Beginners. They are mostly independent learners, unlike the Reluctant Beginners. In general, they work independently, using multiple information sources, trying to learn and work out problems on their own. They tend to seek help from others only when they run into major difficulties.
They tend to be extrinsically motivated to learn programming. These students are willing to put in the effort to learn and understand as best as they can, but are not intrinsically driven to obtain a deeper understanding. For some of these students, their learning task intent is weak. Their main focus is to get the deliverables done to get good marks. These students exhibit some fast tracking behaviours such as not completing tutorial tasks before working on assignments or rushing at the last minute. This results in gaps in their conceptual understanding and can hinder their ability to assimilate techniques that can help them progress their skill level in programming.

“The tutor would give you ten minutes to do the task and then he would actually do it and type it up. And then everyone would look on the projector screen and just copy him. I think the people who enjoyed it would go back and read the prescribed text and try and make sense of it. But people who weren’t motivated, like me, we were just focused on making sure we completed the tasks and getting the marks.” (I11)

There are also those students who try very hard and do all the tasks as required but may not fully grasp some of the key fundamental concepts.

“I have given programming a fair go but it is just one of those things that you either get it or you don’t. I’ve never missed any classes, I’ve gone to help labs and I’ve even seen [The Lecturer] every week. I spend at least two hours every day and on the weekend we would sit there from 10 to 4. I am frustrated with myself with why I cannot understand it. I want to learn! I need to understand what I am learning.” (I4)

The Willing Beginners display an intermediate ownership of their learning. Unlike the Reluctant Beginners, they prefer to exercise their independence by seeking online solutions to problems themselves, prior to doing any face to face information seeking. They show a predominant tendency to prefer online sources such as Google, MSDN and external forums. They may quickly skim their notes or bypass their notes altogether and directly search for solutions online. Teaching staff are consulted as a last resort and mostly at the next tutorial. Very few reported attending help labs.

“When you’re doing your assignment and you get an error code, you look at the book, or Google, but you can’t make sense of it. You can’t connect the two. You just trial and error and guess. So you would do that for 2 or 3 hours. I mean, it is a form of learning but it wasn’t one that would use your intelligence to try and solve a problem.” (I11)
They display ad-hoc Problem Solving Behaviours which include lack of planning and an unsystematic approach to coding and checking errors. They heavily use ad-hoc trial and error in their problem solving. They also have a weak mental model of the inner workings of the program and weak debugging skills. They mostly practice dependent debugging and they rarely use the debugger tool.

“There were some bugs I didn’t understand. But I just changed it, had it fixed. ‘It works! I’m happy about it.’ I did not try to understand the change. I was just apathetic about it. Sometimes I understood what was going on but more or less if it was hard to understand what was happening, I would just try and fix it.” (I20)

The Willing Beginners work mostly co-dependently with their peers and personal networks with either a task or skill division of labour. In their attempt to try solving a problem, the co-dependency interaction with peers usually happens after consulting online sources, such as external forums, but prior to asking teaching staff.

“Once I had it explained to me it did make sense. I was more confident but I wasn’t 100% okay with doing it by myself. Which is when I got friends to help me with some of the code. Mainly the code that loops through, because I wasn’t confident with loops.” (I19)

Six (of 31) participants in this study were identified as Willing Beginners.

### 6.2.3 The Keen Beginner

The Keen Beginners are those students who see programming as highly relevant to their future careers. Some of these students, at this stage, already know they intend to become programmers while others may not have clear goals but do see programming as relevant to their future. Overall, they see value in learning programming and are deeply interested in learning it.

“Now that I have experience with programming, although I’m studying marketing, I probably will be going into programming – or maybe a hybrid. But I see myself more as a programmer.” (I31)

Unlike the Reluctant and Willing Beginners, the Keen Beginners thrive with exposure to programming challenges. This nurtures their interest to grow deeper and improves their independent problem solving skills much more rapidly than the other novices.
The Keen Beginners have a novice Skill Level but higher than the other novice groups and bordering on competent. They are novices because they have just begun to learn programming for the first time. However, they have a deep desire to understand programming problems and are mostly independent learners.

The Keen Beginners have a strong learning task intent. They have a strong curiosity to understand how things work. In their intrinsic interest they are likely to go beyond what is required in their learning in search for a better understanding. They are also highly independent learners. They like to work out problems on their own and come up with solutions to problems independently. They are keen to learn how to solve problems at an implementation (hands-on) level. They are also highly thoughtful when trying to solve coding problems.

“I wouldn’t necessarily check every loop. I’d check the final one and go, ‘Okay, this information here doesn’t line up with all the rest of the information’. And so that’s when I would problem-solve the bug by checking all the loops really thoroughly and find where the data was being lost, essentially.” (I13)

They have a strong ownership of their learning. They tend to seek help only for very challenging tasks in their assignment deliverable. They usually postpone asking questions on an online forum or even to the teaching staff (tutors) until they have tried to work the problem out themselves for a good amount of time.

“You are sort of working by yourself on the assignment and there is not much outside help unless I go and see [My Tutor] ... but it is not one of those questions I want to see him about yet ... I prefer to see him when I am really stuck.” (I5)

Most Keen Beginners have learnt simple debugging techniques, mainly due to their extensive programming practice. Some described learning how to debug as a revelation and a powerful tool. As they are exposed to learning programming, they are more willing to adopt the tips and techniques they observe others use, either from external forums, peers or teaching staff.

They display intermediate Problem Solving Behaviours. They practice simple planning and code step by step. Some may use a thoughtful trial and error approach where they trial and error moving code lines around attempting to understand what the code is doing. They are developing independent debugging skills. In their repertoire of debugging techniques, some may use message boxes or print statements to display the values of variables and to trace the program’s behaviour.
Others rely more heavily on Google and external forums to look at how others have solved a coding problem. Some may even be in the process of learning how to use the debugger and interpret the information it provides.

“For example when I was doing my assignment, we used a loop to read each line from a file. At the start I thought I could put the split outside of the Loop, but that did not work. And I am like ... Why isn't this working? It took me a lot of trial and error because I knew Google would not find that stuff for me. So I played around with it and then I eventually put the split inside the while loop and like yeah!, it worked.” (I5)

The Keen Beginners use all available sources of information but they refer to their completed tutorial work and previous assignments with much more attention. As novices they tend to rely on online sources of information (Google and external Forums) to see how others have resolved a similar error they have encountered. They also look for sample code on how to do something, that will help them with the higher levels of their assignments. Usually they can make sense of what they find online and they are able to take the pieces from the different postings and adapt them to their problem. Some are comfortable asking question on the internal forums but not keen on giving advice to other students.

“I check the lecture notes but I like having another description. I’m normally hearing about it first in a lecture and then if I still don’t understand, I’ll look it up on the web. There are heaps of online communities of people helping each other out. And there are videos from other universities. I’m a firm believer in doing further research online.” (I13)

Most of the Keen Beginners tend to be collaborative independent when interacting with their peers and their personal networks. Often, the Keen Beginners are the ones providing help to the other students. They described enjoying this, because in doing so they were able to come across interesting problems that they may not have encountered themselves, which deepened their understanding.

Similar to the Budding and Advanced Developers, the Keen Beginners learn by looking at what others do on external forums and also by critically comparing their code to their peers. They are also quite reluctant to adopt someone else’s approach unless they fully understand it.

Five (of 31) participants in this study were identified as Keen Beginners.
6.2.4 The Budding Manager

The Budding Managers do not see programming as directly relevant to their future goals, which tend to be towards business analysis and IT management. Although they have a low Perceived Personal Relevance towards programming, some do, however, value that a basic knowledge of programming can make them a better analyst/manager.

“I wanted to be a business analyst. I definitely reckon it [programming] would come in handy. Because meeting the client’s needs is hard if you don’t know how. You have to tell them something that can be done, that’s achievable.” (I26)

They have a good level of competency with programming basics, but they are keen to learn how to solve problems at a higher abstract level. They tend to not be interested in the technical, hands-on implementation of the solution and admit to steering away from any involvement in development. The students in this profile have a strong sense of time and how long things take to learn, especially with programming.

“The bit that I enjoyed most was being given a problem and having to work your way through it. The bit where you see how your work has unfolded, I see the value of that. I was happy to do a little bit of programming, but I didn’t want it to be the main focus. I enjoyed the analysis more than the core programming. More the high level: what is the problem and who are the customers? Stakeholder analysis, that sort of thing.” (I10)

Budding Managers have a mixed Learning Trait. They are generally independent at the basic levels of programming tasks but they become dependent when things get hard, beyond the point of what they see as relevant to them. When trying to achieve higher or advanced levels of tasks takes longer than what they expected, they become co-dependent on others, relying on their preferred face-to-face sources of help (e.g. help lab tutors, peers and personal networks). Some may even display a one-way dependency, especially when deadlines are approaching.

“When you’re under pressure and you don’t have the time to do everything… in the study group we were all doing the same task … if one person didn’t have the code, another person might have the solution for something else … I tended to give up quicker, because you just wanted it to make it work.” (I26)
“At first I got frustrated with myself, but after a while I just accepted it and I went to all the help labs. I got all the extra help I could get. I thought ‘I am going to give myself the best chance to do it’, and I got brilliant marks in all my programming subjects, but I think it was that sucking it up and going to those labs that helped.” (I16)

These students are primarily extrinsically motivated to learn programming. Their learning task intent is mixed. They are motivated to learn the basic concepts that they perceive will help them become better analysts/managers. They do, however, tend to be achievement motivated - obtaining good marks is important for them. They are not inherently interested in learning programming for its’ own sake.

“Unfortunately I wasn’t one of those people that wanted to do it in the best most efficient way - I just wanted to get it done! When I really understood it then I really got excited about it, but as soon as it was something that I was struggling with and I found it really difficult, I just wanted to get it done - I didn’t care”. (I16)

All the students in this group have learnt to make good use of their personal networks to achieve their goals, more than the other profiles. They tend to work in study groups and leverage the strengths of others but also contribute to the group. They may interact co-dependently with their peers and personal networks with planning and debugging using skill based division of labour and some will genuinely learn from their co-dependent interactions with their peers/personal networks. With coding they may use a task based division of labour.

“I was surrounded with my friends … they’d done Programming. So it was a very collaborative effort from all of us. There was a lot of team dynamics with the way we worked. For example, my cousin was very quick to put a skeleton of a program together. And then I was very quick to actually embellish on that.”(I17)

They display an intermediate ownership of their learning of programming. When the tasks are viewed as relevant, they will genuinely have a go and try to work through problems themselves, prior to seeking information from their personal networks. However, for advanced topics that they find difficulty in understanding immediately, their ownership weakens, giving way to co-dependency on others.

Some of these students may still work on assignments with incomplete or misunderstood tutorial work and take an ad-hoc trial and error approach to solving coding bugs. Their debugging skills are still not developed and they rely heavily on their peers and/or personal networks for help with debugging.
“We reviewed what the task was and what we thought was the way to go … and then sort of try and find the bits in the notes that are probably applicable to solving it. There wasn’t a thorough plan - just trial and error.” (I10)

“I was like, ‘there’s a bug there! There has to be a way to fix it’. So it was like a very collaborative way to learn off each other. Even up until now, I know how to debug programs very well. A bit on my own, and a bit with my friends … and the teacher helped to just guide that along”. (I17)

The Budding Managers use and pay considerable attention to all their sources of information. However, they show a predominant dependency on their peers and personal networks when seeking help and information for planning, coding and debugging.

Most described using external forums as an information source, primarily as an on-line manual looking for syntax examples and sample code to guide them along. Most will check external forums prior to consulting their peers and personal networks. They do, however, rely more on their peers and personal networks for solving coding problems. Often when they do find instructions on external forums, they find them difficult to follow and implement without the help of their peers and personal networks.

“I Googled a lot, but didn’t understand what they were trying to explain because they were on a different level of what I was… I read a lot of forums, especially Stack Overflow. They would show code and other people would just input what they thought. I followed the instructions, but it didn’t work for me.” (I26)

Four (of 31) participants in this study were identified as Budding Managers.

6.2.5 The Budding Practitioner

The Budding Practitioners are those students who see programming as relevant to their future career goals but do not intend to become programmers. They are interested in taking on technical roles (such as Business Analysts, Database/Network Administrators or web developers). They see value in learning programming in order to become better IT practitioners. Some also have an interest in programming as a hobby or for personal development but do not want to do it as a career.

“Programming is something I would consider as a pathway but it is not my primary pathway. I don’t mind doing it for a while, to practice and understand systems, but my ultimate goal is to be a business analyst.” (I2)
Unlike the Budding Managers, the Budding Practitioners are enthused by technology and are interested in solving problems using technology. They are comfortable working at a hands-on level more than the Budding Managers and actually enjoy doing development and don’t mind doing it. Their focus is on learning the programming language or technology that will be useful to them for, say, developing a customer end product.

“Well I definitely enjoyed learning programming but my focus is to help my parents who want to start a business which they’re keen on.” (I22)

The Budding Practitioners are competent programmers and have a good understanding of programming concepts. They are mostly independent learners, unlike the Budding Managers. They make extensive use of external forums, for their perceived speed of response, as a trusted source and also to learn from. They are confident with the keywords they use in Google searches to help them find relevant sample code or examples of how others have fixed an error. They generally understand what they find in the forums and also are able to follow the recommended steps. They have a good mental model of their code and of the problem they are trying to solve. Face to face assistance from teaching staff is left to when they are really stuck. Most did not report needing to attend help labs.

“Well it’d be faster because you’d look up the error, if someone’s already asked the question and received an answer, you can read what they’ve said.” (I22)

Their motivations are extrinsic because their focus is learning the technology that will help them develop an end product rather than the programming itself. They are not particularly concerned with the efficiency or style of coding – they will use whatever works. They are open to learning mild forms of programming such as HTML for web development, scripting for networking/databases and SQL.

These students have a strong learning task intent. They are equally interested in completing task deliverables to achieve good marks as well as having a good understanding of concepts. Although they may sometimes take some short cuts they have a strong ownership of their learning of programming. They want to genuinely understand and work out problems. Even when they consult how coding problems were solved, they are critical and learn by comparing what they did with what their peers and/or personal networks have done. They are quite proactive in trying out different solutions to choose the best option.
These students have intermediate to strong Problem Solving Behaviours and improve as they are exposed to programming challenges. Initially they may use trial and error (for example, re-writing code from scratch to wipe out problems), however, as they progress, they become more thoughtful about what they do.

“Probably at the beginning of each semester you trial and error ... as I learn the language I try to think more about it” (I2)

They proactively work out their coding problems independently. They predominantly consult external forums, prior to consulting their peers and/or personal networks, after they tried various alternatives. They critically compare what they have done with their peers and learn from the comparison. Their descriptions of how they use the programming debugger tools as an information source is basic, however.

“First I would have a look at the complexity of the problem. If it’s just a simple typo, I quickly changed it but if it is something that looks confusing, that is when I would go to forums to find out if anybody else has figured out how to go about it. If there were no posts, that is when I actually make the post and some person would reply.” (I2)

They work collaboratively independently with their peers and/or personal networks (for planning and debugging) or independently. Implementation and most of the debugging is done independently. Their planning and debugging is not as evolved as the approach used by the Budding and Advanced Developers.

Three (of 31) participants in this study were identified as Budding Practitioners.

6.2.6 The Budding Developer

The Budding Developers are those students who see programming as highly relevant to their future career goals as they are considering doing some form of programming in their work. They see a high value in learning programming as they love using IT, especially programming, to solve business problems. They are deeply interested in learning it and are willing to put in as much effort as is necessary to learn the intricacies of the craft. They are more intrinsically interested in learning programming than the Budding Practitioners.
Their interest goes beyond using programming and / or the technology to solve a business problem. They are also strongly interested in finding the most efficient approach (coding style and structure) and in creating the most elegant code they can write. Their planning is also more thoughtful and they have a stronger mental model than the Budding Practitioners and Keen Beginners. They are starting to think about modularity and code architecture.

“I think I had originally something like probably about 110 different forms. [Since] the start of this semester ... I sort of looked at [my software] and I’m thinking, ‘This is way too much.’ Learning about functions has helped me to reduce my program by almost 50% in terms of forms and reusing code. It’s so much more efficient.” (I18)

They have the same intrinsic motivation as the Keen Beginners but have advanced their skills to a competent level. Also, they have a deeper understanding of programming concepts and are more competent in code writing and debugging than the Budding Practitioners. Not only do they go beyond the scope of their studies, many do development outside their studies for family, friends, local sporting and community clubs.

“When I finished the first programming course, I started to design my own point-of-sale software for my local sporting club.” (I18)

These students have a strong Learning Task Intent. They are strongly interested in understanding programming concepts. Completing the tasks is a vehicle for exploration and reaching that desired understanding. But also, they are strongly interested in going beyond the scope of what they are learning. They have a deeper curiosity than the Budding Practitioners and Keen Beginners.

“Even if I didn’t have an issue, I was curious to see what other things are out there. I would be watching videos of lectures from other schools to learn as much as possible. I didn’t program on the side but I definitely studied it on the side. I would watch videos about basic programming, basic language, how to attack a problem, the thought process. I watched heaps of videos on HTML, PHP, and XML.” (I15)

“I want to learn more languages, even if it’s at a base level. On semester break I started learning JavaScript. Some of that stuff is a bit new to me and it’s a base level. So I definitely have a desire to want to learn more of them. I was talking to my tutor that I’d gone on the Code Academy website and was thinking about learning Python.” (I29)
Similar to the Keen Beginners, they have a strong ownership of their learning. They are independent learners more so than the Budding Practitioners and their ownership is stronger. They are likely to try working things out on their own, using multiple sources of information, for a longer period of time, before resorting to asking questions on forums or from peers, personal networks and teaching staff.

“Depending on the degree of the problem, I always try to fix something myself first. I wouldn't just want the answer. I want to fix it myself. I want to solve the problem because, as I said previously, that is the way you learn it.” (I7)

They generally refer to all sources of information with more attention. With more programming maturity and experience, they are more likely to keep going back to their notes and completed tutorials more than the Keen Beginners and Budding Practitioners.

“In the programming class, what was in front of you was relevant to the task at hand, so as long as you knew that information it was easily adaptable to the final product. That is the beauty of it! That is why less time is spent on it because it is there in front of you. Whereas for example in Accounting theory, you read a book that is quite thick and you are not too sure of what you specifically have to focus in the book.” (I7)

“If it was a tutorial thing, I would go back to see if I had missed something in the tutorial notes. Actually, I've done this twice now this semester. If it doesn’t look like what it’s meant to be because I’ve skimmed / missed something that's important.” (I29)

They are avid users of external forums, mostly to look at preliminary information when working ahead, or additional information that is complimentary to what is in the notes. They can understand what they find on the forums and can follow the recommendations. Furthermore, they use forums to learn better ways of doing things (i.e. code writing).

“I probably used the Internet more for the website stuff like using JavaScript and PHP, just seeing how people are setting it out, I guess, in a way. What the teachers were telling us and showing us was definitely sufficient to get things done. But I just wanted to get as good as I could.” (I15)

Few of these students are likely to use the internal forums before thoroughly consulting external forums. Only some of these students are confident enough to post on internal forums, mostly because of concern of being judged by their peers.
The students in this profile have thorough Problem Solving Behaviours. They do thoughtful planning and error detecting. They use desk checking to understand what their code is doing and the values the variables should have. They practice efficient and independent debugging and are comfortable using the debugger and all the features it provides. As well as the debugger, they still use all the techniques practiced by the Keen Beginners such as print statements, thoughtful trial and error and searching Google and external forums. These students have a good mental model of the execution flow of their program which helps with isolating the problem area.

“I used the de-bug tool and stepped over, used break points, watches, and then clicked on the dropdown boxes on it, to see what was stored in the variable array list. That made it a lot easier, to actually see how it was forming. I also used message boxes. They are not as difficult but they are very tedious and printed to the console.” (I15)

The Budding Developers are mostly Collaboratively Independent when interacting with their peers and personal networks. Some, however, may be Solitary Independent. They consult their peers after they have done it themselves and they learn from the comparisons. As they have a strong ownership of their learning, they are not keen to adopt their peers’ approach unless they understand it and believe it is better.

“This year there’s a few class mates that I would work together with. We message each other or email each other. Sometimes their way of thinking is slightly different to yours and you still will get the same answer but you’ll do it differently to them. If you’re really stuck, you talk to them in class, ‘How does yours looks? This is what mine looks like. Where am I going wrong?’” (I29)

Four (of 31) participants in this study were identified as Budding Developers.

### 6.2.7 The Advanced Developer

The Advanced Developers are similar to the Budding Developers in that programming is highly relevant to their career goals and they have a deep interest in learning it. They have, however, advanced their skills to a higher level, mainly through their extensive practice in solving problems through programming. They are intrinsically motivated to learn all aspects about programming but are also strong independent learners.
In this study, all the students that were classified as Advanced Developers exhibited both a high Perceived Personal Relevance and also an independent Learning Trait. These two dimensions seem to be vital ingredients in advancing programming skills to that of an Advanced Developer.

“I spend days upon days trying to figure it out myself, Googling, searching things on forums. This is before having done any structured learning. I enjoy being able to create something that was functional through coding. Understanding the problem and the planning phase were probably the most enjoyable. I like doing that sort of thing but it is more enjoyable to see the results working at the end.” (I7)

They have reached the mindset of a professional developer in that they display a more mature and thoughtful approach compared to the Budding Developers. They not only focus on the coding but, most importantly, on code architecture and code flexibility which anticipates likely future customisation (changes in user requirements). They are better and faster in most of the programming phases including thoughtful planning and error detecting. They have a stronger mental model not only of how their code is meant to execute but of the code architecture they are using.

“I like Programming because you have to think, before you start, you have to plan whether it’s going to be expanded, or it’s going to be just that system. Yeah, that’s what I like about it. It’s like an art, like each person has their own style of writing.” (I12)

They also display a highly positive emotional engagement with programming. They not only enjoy it, they love it! Similar to the Budding Developers, most of these students are already actively programming in the real world while doing their studies. Even the ones who have not yet experienced real world development still have the same approach and thinking.

“There’s a guitarist that I really like, and he said once, ‘I would have given it up a thousand times if I didn’t love playing the guitar.’ I loved the programming courses. I would look forward to going to class. I couldn’t wait to start the assignments, and work on them. Yeah, so in terms of my experience, I loved every second of it.” (I8)

“When I watch TV, suddenly something might come up in my mind, and I just go and do it and once I already started, I can’t stop. When I look at the clock, it’s already 10 o’clock and I missed my dinner. Once you start, you have to stay up until you’re done especially if I stumble on a problem, I can’t go to sleep.” (I12)
These students have a very strong Learning Task Intent. Like the Budding Developers, they like to explore in order to reach a deeper understanding. Not only do they complete all their tasks but also do additional tasks that they may find online to satisfy their curiosity and also to go beyond the scope of what is expected in their learning.

They have a very strong sense of ownership of their learning of programming. When trying to solve very challenging tasks they try independently for extended amounts of time, more so than the Budding Developers. In many cases, when assistance or guidance/direction is required, it is with advanced issues.

“By doing more, you learn more about the software and you learn more about the coding too, to go above and beyond. We are here to learn and not just do what you have to do. I want to get something out of it!” (I7)

The Advanced Developers tend to mostly be Collaborative Independent when interacting with their peers and personal networks although there may be some who are Solitary Independent. The Collaborative Independents learn by looking at what others do. This may be on an external forum but they also learn by helping others solve their coding problems. In this interaction with their peers, they come across problems they would not have thought of and by helping out their peers, their learning is enhanced. The Advanced Developers tend to be the student tutors that help the other students.

“My friends and I, we would work together. We’d help each other if we had problems. We’d help debug things, ‘Well, I think your problem’s here,’ or, ‘you’re writing this wrong’, which was very different to a lot of the other students in the class.” (I8)

The Advanced Developers thoroughly understand and go through all their written material. If they consult forums it is to research things they absolutely know are not covered in their notes. Similarly to the Budding Developers, these students use external forums mostly as an online tutorial that can satisfy their strong curiosity with programming, such as learning new programming languages independently and to advance their ways of doing things. They are comfortable reading and posting questions both on the external and internal forums, often checking for posts on the external forum prior to asking on the internal class forum. Also, some of the students in this profile described posting help to other students on the internal class forum.

“So I do my own study more in depth, of a topic. Basically what comes up in my mind, I just type it. There are a lot of websites. I mostly visited StackShare and there are a lot of examples. I find I have to look at which one is suitable.” (I12)
The Advanced Developers display thorough Problem Solving Behaviours. They are well practiced and intuitive in the way they use the debugger as a diagnostic tool. Their reading and tracing the program execution is faster than the Budding Developers. They can make sense of the debugging errors as well as narrow down the problem area in their code as they have a very strong mental model of how their program should execute. Some described looking up the meaning of debugging errors specifically for programming languages they were learning independently.

“I am a thinker. I will sit back and really envision what I’m trying to get to. So I would really put myself into it to try and work out where the problem sat, and really try and get down to where the problem was occurring, to try and fix it.” (I14)

Four (of 31) participants in this study were identified as Advanced Developers.

6.2.8 Summary

Seven distinct Programming Learner Profiles were developed during the analysis: Reluctant Beginner, Willing Beginner, Keen Beginner, Budding Manager, Budding Practitioner, Budding Developer and Advanced Developer. Each profile describes the nature and behaviour of different types of students.

The Reluctant Beginners are those students who do not see programming as being relevant, at all, to their future. These students go through the learning of programming as something that needs to be done but have no real interest in learning it. Not all Reluctant Beginners are first year students. They are highly dependent on others for their learning and mostly prefer face-to-face interaction with their helpers as they find it difficult to make sense of notes or code they find online. They also skip learning tasks and move on to assignments without understanding previous concepts.

The Willing Beginners are those students whose career goals are wide ranging and they do not know if programming will be relevant to them. Their Perceived Personal Relevance of programming is low but they are still open to learning it. While some experience an increase in interest level, others lose interest when concepts become hard. Willing Beginners are extrinsically motivated to learn programming and not driven towards seeking a deep understanding. They practice fast tracking behaviours and skip some learning tasks. They are mostly independent learners, referring to multiple sources but preferring online sources (forums), consulting peers and staff for major difficulties. They work co-dependently with their personal networks, with either a task or skill division of labour.
The Keen Beginners are those students who see programming highly relevant to their future careers. Some already know they want to be programmers. They see value in learning programming and are deeply interested in learning it. They thrive with exposure to programming challenges which helps them develop their independent but also thoughtful problem solving skills more rapidly. They have a strong curiosity and seek a deep understanding usually going beyond what is required. They are keen to learn how to solve problems at a hands-on level. They readily pick up problem solving techniques from multiple sources and personal networks. They can make sense of the sample code they find online and can adapt different pieces from different postings to their problem. They work in a collaborative independent fashion and learn by critically comparing their code to that of their peers and from external forums.

Budding Managers do not see programming directly relevant to their career goals, which tend to be towards business analysis and IT management. They are keen to learn how to solve problems at a higher abstract level steering away from any involvement in development. These students are independent at basic levels of programming but become co-dependent on others when things get hard specially for concepts they do not see it is relevant to them. Their skills are at a basic level but they tend to have achievement motivation and so they manage to achieve their higher goals by their ability to make good use of their personal networks. Their debugging skills are still not developed so they rely on a co-dependent skill division of labour for more complex coding problems.

The Budding Practitioners see programming relevant to their future career goals viewing it as a stepping stone to become better IT practitioners, but do not intend to become programmers. They have an interest in obtaining a good understanding of programming concepts with the focus being on the end product and the technologies used rather than the programming. They work collaboratively independently with their personal networks, critically comparing and learning from each other. They are confident using external forums as they can make sense of the information they find.

The Budding Developers see programming as highly relevant to their future career goals as they consider doing some form of programming work. They see value in using programming to solve business problems. They are intrinsically interested in learning programming and the intricacies of the craft. Their planning, coding structure, debugging and problem solving is more thoughtful and thorough. They have a deeper curiosity to explore and go beyond what is required. They use external forums, comparing and contrasting multiple sources of information to find better coding approaches. They work Collaborative Independently with their personal networks, with some being solitary independent.
The Advanced Developers, similar to the Budding Developers see programming highly relevant to their future career goals but they have advanced their programming and problem solving skills to a much higher level. They have reached the mindset of a professional developer where their focus goes beyond the coding, considering issues of code architecture and flexibility which anticipates future customisation. They are strong independent learners. They not only enjoy programming, they love it. They also actively program in the real world while doing their studies. They work collaboratively independently or solitary independently but they also learn by helping others. They use external forums mainly for their independent learning that goes beyond the scope of what they are learning. These students are the most likely and confident to post help to other students on the internal forum.
6.3 Pathways of Student Learning

During the analysis some trends were identified from descriptions students gave of their past and present behaviours and experiences. It became apparent that there were some pathways that students tend to follow in relation to moving from one profile to another. As these observations are based on the discussions of only some of the students – those who were in the later years of their degree or graduates - a complete analysis of the pathways of all the students was not possible.

The more prominent pathway progressions are presented in this section to give a potential picture of how the profiles relate to each other and how students may evolve from one profile to another.

Keen Beginner -> Budding Developer -> Advanced Developer

The most obvious path progression observed was that of Keen Beginners becoming Budding Developers and then perhaps moving up to Advanced Developers. It was quite evident, based on the recollections Advanced Developers gave of their experiences when they first started their degree, or previously in high school, that they all started as Keen Beginners.

“In early high school, I started messing around with HTML. I remember being really excited about getting a Javascript pop-up box to come up. I thought, ‘this is awesome’, and that’s when I really started to take more of an interest in programming. When I realised I could actually make software and make it do exactly what I want, I think that was the moment I thought ‘Oh wow this is amazing what I can do with this’” [18].

For some Advanced Developers, the move up to Budding Developer occurred towards the end of their first course while for others, who were still developing their skills, it occurred towards the end of their second or even third programming course or their co-operative year education.
The discovery of how to debug coding problems and how to use the debugging tool as an information source was a major turning point for these students. Learning how to work through bugs and being able to check and trace problem code as it is executing was described as a breakthrough moment. The practice and mastery of debugging skills gave these students the confidence and freedom to spend more time on other aspects such as building a program to solve a problem in IT or a focus on architecture and the way code can be written with a more efficient structure.

“I think once I learnt how to use the debugger it was immensely helpful.” (I3)

**Reluctant Beginners & Willing Beginners -> Budding Managers**

Budding Managers tended to start as Reluctant or Willing Beginners. In their recollections of when they first started their degree, they all described a low Perceived Personal Relevance towards programming. The Reluctant Beginners moved towards Willing Beginners as they began to understand the basics and improved their skill level somewhat, as well as being achievement motivated. These students moved to Budding Managers by learning to leverage their student personal networks to improve their learning ability and achieve higher grades. Budding Managers tended to emerge in the second year of their degree.

“To be brutally honest with you, I thought it was quite boring. If you didn’t understand the basics of it, it was quite hard. After a while it became repetitive and gradually became easier, and I started enjoying it after a while, when I got my head around it and I learnt to read the code properly.” [I27]

**Willing Beginners -> Keen Beginners -> Budding Managers**

An alternative pathway for some Budding Managers is that of starting off as Willing Beginners and moving up towards Keen Beginners during their first programming course. These students have a more evolved skill level with a higher Perceived Personal Relevance towards programming and a sense of enjoyment as they begin to understand the basics of programming. However, in the second programming courses as concepts become more complex, they trend towards Budding Managers instead of Budding Practitioners or Budding Developers when they lose interest in dealing with more complex programming problems.
"After doing the programming subjects and then going on to design my own stuff, I got a real interest in it because I thought, ‘Well you know what? I can do it in here. I’m sure if I apply myself, I could do it outside.” [I18]

Willing Beginners & Keen Beginners -> Budding Practitioners

Budding Practitioners in their learning pathway tended to start as either Willing Beginners that moved towards Keen Beginners and then moved up to Budding Practitioners or they started directly as Willing Beginners who then stepped up to Budding Practitioners. Those Willing Beginners who have a slightly higher Perceived Personal Relevance towards learning programming than the other Willing Beginners, tend to move up towards Keen Beginners as their programming skills develop. This step up is likely to happen towards the end of their first programming course.

When Willing Beginners move up to Keen Beginners, their interest in programming also develops as they become more open to some form of mild scripting. The move from Keen Beginner to Budding Practitioner is likely to occur either towards the end of their first programming course or in their second year of their degree as the students learn IT related courses that include mild versions of scripting.

The Keen Beginners who move up to Budding Practitioners instead of Budding Developers are those students who have a lower Perceived Personal Relevance towards programming but see it as an important stepping stone towards their future career goals. Also, the students who step up towards Budding Practitioners described having a higher interest in IT and IT related fields but not necessarily with programming.

“Programming is something I would consider as a pathway but it is not my primary pathway. I don’t mind doing it for a while, to practice and understand systems, but my ultimate goal is to be a business analyst.” (I2)

On the other hand, the Willing Beginners who move up directly towards Budding Practitioners are those students who start off with more evolved skill level but have a lower Perceived Personal Relevance towards programming compared to those Willing Beginners who move up towards Keen Beginners.
No Progression

Some students do not progress beyond the profile they begin their studies with. For instance, it was evident that some Reluctant Beginners managed to get through their entire degree without moving beyond the Reluctant Beginner stage. Even some students that had graduated admitted that although they passed their programming courses, they did not really understand everything they learnt and were not confident to do programming in industry.

“If you ask me about the theory I can tell you about it but then I am still not confident enough in the practical work, even at this stage.” (I1)

“I always thought programming is something very difficult. Personally I am not keen to be a programmer so maybe I did not work hard enough to be a programmer. I found some other areas such as USE CASE analysis very interesting.” (II)

Some Reluctant Beginners may move up towards Willing Beginners with programming and remain Willing Beginners with regard to programming for the rest of their degree. However, they may step up towards Budding Managers or Budding Practitioners in the other IT related areas such as networks, databases or web development. It is possible that a Willing Beginner may find programming very hard and having a very low Perceived Personal Relevance, they can step down to a Reluctant Beginner.
6.4 Relationships

Programming Learner Profiles bring together the many influences that arose during the analysis, in a unified and cohesive manner to better understand them. This section focuses further on these influences by describing the various influential relationships that evolved during the analysis and development of the model.

Section 6.4.1 describes the bi-directional relationship between Learner Nature and Learning Behaviours.

Section 6.4.2 discusses the relationships within Learning Behaviours: the interrelationships within Core Learning Perspectives and how core learning perspectives influence Patterns of Collaboration and Patterns of Information Use. Section 6.4.3 presents and discusses a table describing the collaboration preferences of each Programming Learner Profile, highlighting the influences that the profiles have on Patterns of Collaboration and Personal Networks. Section 6.4.4 then presents and discusses a table detailing the information behaviour preferences of each Programming Learner Profile, highlighting the influences that the profiles have on Patterns of Information Use and Information Sources. Each of these sections gives a different perspective on the complex interplay of influences that impact the programming learning experience.

6.4.1 Learner Nature Relationships

There is a bi-directional influential relationship between Learner Nature and Learning Behaviours. The stronger relationship is the influence that the concepts within Learner Nature have on Learning Behaviours. While Perceived Personal Relevance, Learning Trait and Skill Level all have an influence on learning behaviours, Perceived Personal Relevance seems to be the strongest. This section describes how Perceived Personal Relevance influences: Core Learning Perspectives, Patterns of Collaboration and Patterns of Information Use. How Learning Trait and Skill Level influence Learning Behaviours is also discussed.
Perceived Personal Relevance Influence on Core Learning Perspective

Perceived Personal Relevance was found to be highly influential on the Core Learning Perspective concepts, primarily Ownership of Learning and learning task intent but also with Problem Solving Behaviours.

Students who perceive that learning programming is relevant to their future described having a deep desire to thoroughly understand their learning tasks and deliverables. They are intrinsically interested in learning it so they willingly do their own exploration in order to seek understanding, so they can use what they learn in the future. They perceive that they need to develop their skills in order to tackle similar problems in the future independently, which increases their level of ownership of their learning and also their Problem Solving Behaviours.

On the other hand, the students who do not perceive programming as relevant to them are extrinsically interested and are more inclined towards completing the course deliverables rather than learning the concepts. As they do not see themselves as future programmers, their Ownership of Learning is lower and their learning task intent is primarily focussed on getting the tasks done and moving on. They would rather spend their time and effort on subjects that they have a higher Perceived Personal Relevance. Their Problem Solving Behaviours will remain weak as less effort is spent on developing these since mastering techniques like debugging are considered irrelevant to their future career goals.

Perceived Personal Relevance Influence on Patterns of Collaboration

Students with a higher Perceived Personal Relevance tend to interact with minimal reliance on others. They primarily work independently but collaborate and share knowledge with their trusted personal networks in order to achieve higher understanding and learn from each other. Similarly, Solitary Independent students work independently but interact with their personal networks primarily as help providers and rarely as help seekers.

Students who reach this independent and collaborative level of interaction with their peers and personal networks will tend preserve these styles of interaction even when things become complex.
On the other hand, the students who have an intermediate or lower Perceived Personal Relevance with their learning of programming interact with higher reliance on others. Some students who primarily interact with a One Way Dependence are likely to remain at this level of interaction. Other students who may progress towards a Two Way Co-dependent interaction, some of which may stagnate at this level if their Perceived Personal Relevance remains the same or may even revert back to a One Way Dependency interaction if what they are learning is perceived complex and not relevant to their future career goals.

**Perceived Personal Relevance Influence on Patterns of Information Use**

Students with a higher Perceived Personal Relevance tend to have a strong Ownership of Learning and a strong learning task intent, as they see this learning as relevant to their future. They also progress towards practicing thorough Problem Solving Behaviours more rapidly. These improvements also have an influence on their Patterns of Information Use in that as their skills improve, they also improve the way they use their information sources. They use their programming tools more efficiently for syntax verification and are more able to find the meaning of errors, see how others have fixed a problem on forums and then fix their coding problem. Most importantly, they move towards practicing the more evolved solution discoveries of code adaptation, comparative analysis and beyond the scope.

Students with a weak Perceived Personal Relevance tend to have a weak Ownership of Learning although there may be some students who have a strong Ownership of Learning. They also tend to have a weak Learning Task Intent primarily seeking to get the task done and only seeking a basic understanding of the areas they perceive may be relevant to their future careers. Their Problem Solving Behaviours tend to be mostly weak. Very little effort is spent developing debugging techniques to fix coding problems or spend the time working through coding problems as these have a low Perceived Personal Relevance. These Core Learning Perspectives influence how these students’ use of information sources in that they will use their information sources mostly as basic references but only practice some of basic solution discoveries such as looking for sample answers. As these students have weak Problem Solving Behaviours they would find it difficult adapting the sample answers they find using Google. Their tendency is to look for complete answers.
There are also some students who have a higher Perceived Personal Relevance but for personal reasons such as working commitments, family reasons or personal issues, are unable to dedicate the time to develop their skills. These students tend to adopt an intermediate Ownership of Learning, assuming partial responsibility for their learning, however some may still have a Strong Ownership. They also tend to take short cuts and adopt an intermediate Learning Task Intent where they seek some basic understanding but mostly they try to get their learning tasks done. Their Problem Solving Behaviours tend to also be intermediate. These Core Learning Perspectives influence how these students’ use of information sources in that they will use their information sources as basic references but only practice some of basic solution discoveries such as looking for sample answers. They may also practice some code adaptation as well but will not practice the more evolved solution discoveries.

**Learning Trait Influence on Learning Behaviours**

Learning Trait has a number of influences on Learning Behaviours. Firstly it is an abstracted form of Patterns of Collaboration, but it also influences Ownership of Learning, Problem Solving Behaviours and Patterns of Information Use. Learning Trait is an abstraction of Patterns of Collaboration that describes how students interact with others to work through problems. This higher level form was used to define the profiles.

Patterns of Collaboration, on the other hand, provides a more detailed and richer description of how students tend to work through problems with and through others. Patterns of Collaborations were used to compare and contrast the profiles. Students who are dependent work in a One Way Dependency or Two Way Co-Dependency interaction with others whereas students who are independent work in a collaborative independent or solitary independent fashion. They display minimal reliance on others but help others learn and also they learn by helping others.

Learning Trait influences Ownership of Learning and also Problem Solving Behaviours in that students who tend to have an independent attitude, have a stronger ownership of their learning as they want to assume responsibility for their learning and learn how to resolve coding problems by themselves. Students who tend to have a dependent attitude, have a weaker ownership of their learning, believing others are responsible for their learning. They are more passive in the way they learn and resolve coding problems.
Learning Trait also influences Patterns of Information Use. Independent students tend to prefer to resolve coding problems by comparatively and progressively working through their notes and online forums for an extended amount of time. They seek to only consult others through collaboration by comparing their approach after they have tried it themselves. They seek to consult teaching staff as a last resort. Dependent students, on the other hand, highly prefer to consult with others face to face, preferring an active demonstration of how to work through a problem step by step. Online forums and tutorials are not used to their fullest potential as these students find it difficult to make sense of the information they have found and implement it in their coding.

**Skill Level Influence on Learning Behaviours**

Skill Level influences Learning Behaviours as a whole by influencing the concepts that make up Learning Behaviours - Patterns of Collaboration, Core Learning Perspectives and Patterns of Information Use. Skill Level influences Patterns of Collaboration in that the students who are more highly competent and advanced in skill level predominantly interact in a Collaborative Independent fashion and also help others. The students that are less skilled tend to interact in a One Way Dependency or in a Two-way Co-dependency where they also depend on others completely or on a particular skill.

Skill Level also influences Problem Solving Behaviours. The more highly competent and advanced students predominantly practice intermediate and thorough Problem Solving Behaviours. They work independently and carry out a thoughtful and proactive approach to coding problems in the way they plan, code and work out the problem triggers in their code. The less skilled however, display ad-hoc Problem Solving Behaviours where they skip tasks, jump straight into coding without a plan and have an unsystematic coding approach and dependent debugging.

Skill Level influences Patterns of Information use because only the more highly competent and the advanced skilled students practice the more evolved solution discoveries (Code Adaptation, Comparative Analysis and Beyond the Scope). The less skilled students, on the other hand, practice syntactical code verification and looking for meaning of errors, but are unable to find the problem trigger and when looking for sample answers they mostly search for complete sample code.
6.4.2 Learning Behaviour Relationships

The concepts that define Core Learning Perspective are closely related and intertwined. These concepts, in turn, have a major influence on the Patterns of Collaboration with others and the patterns of how they use their information sources. As learning behaviours evolve, these can also have a feedback influence on Learner Nature.

Core Learning Perspective Interrelationships

Ownership of Learning, Learning Task Intent and Problem Solving Behaviours are closely related and intertwined. In general, students who have a strong Ownership of Learning also have a strong Learning Task Intent and practice thorough Problem Solving Behaviours. Similarly, students with a weak Ownership of Learning tend to also have a weak Learning Task Intent and Problem Solving Behaviours.

In some cases, students may have mixed Ownership of Learning, Learning Task Intent and Problem Solving skills. Novice students that are still developing their skills may have strong Ownership of Learning and Learning Task Intent but still practice weak Problem Solving Behaviours. Other students may have a will to learn and have a strong Ownership of Learning but may be constrained by their circumstances outside of study. Students who are working full time, or looking after a family, may not have the time to devote to their learning tasks and develop their problem solving skills.

The students’ attitudes toward their learning responsibility influences their intentions towards their programming learning tasks - whether they want to conceptually understand a problem or just get the task done. These intentions will influence what the students do and how they approach it: their behaviours, actions and steps they take when they problem solve.

Core Learning Perspective Influences on Patterns of Collaboration

A stronger Ownership of Learning, with a Learning Task Intent inclined towards understanding along with more thoughtful and thorough Problem Solving Behaviours will influence a Pattern of Collaboration that is Collaborative Independent or Solitary Independent. The strong Ownership of Learning these students assume influences a strong desire to understand by themselves not only the coding problem but also what method was used and how it was used for solving the problem.
These students are proactive in the way they problem solve with their personal networks. They are independent in the way that they approach their problems but also they collaborate with their personal networks by comparing what they have done or how they solved a problem to others. This comparison of approaches with others enables them to learn from the differences and find better ways of doing things. Also they learn by helping others.

A weaker Ownership of Learning with a Learning Task Intent inclined towards getting the task done to achieve the marks, along with ad-hoc Problem Solving Behaviours will influence a Pattern of Collaboration that is passive and dependent on others. The weak Ownership displayed by these students influences their reliance on others as they see their learning is the responsibility of others. They display a weak Learning Intent forgoing the desire for deep understanding and this has an influence on their Problem Solving Behaviours.

Their Problem Solving Behaviours are weak in that they display ad-hoc and unsystematic problem solving usually skipping steps such as thinking and planning or learning how to solve a coding problem independently. These short cuts in the way they problem solve is revealed through their passivity in mastering how to solve coding problems and this influences a One Way dependent to Two Way Co-Dependent pattern of collaboration with others, especially for how to go about coding (planning) and predominantly solving coding problems (debugging).

Core Learning Perspective Influences on Patterns of Information Use

A stronger Ownership of Learning, with a Learning Task Intent inclined towards understanding, along with more thoughtful and thorough Problem Solving Behaviours, will influence a Pattern of Information Use that is more evolved, allowing the student to be less reliant on complete answers and be able to identify, adapt and evolve code through comparative analysis and go beyond the scope.

A weaker Ownership of Learning with a Learning Task Intent inclined towards getting the task done to achieve the marks, along with ad-hoc Problem Solving Behaviours, will influence a Pattern of Information Use that is reliant on complete answers.
Learning Behaviour Influences on Learner Nature

There is also a feedback influence that occurs between Learning Behaviours and Learner Nature. As students develop and improve their Learning Behaviours, Patterns of Information Use and Problem Solving Behaviours, these will naturally improve their Skill Level. As their Skill Levels improve, students will progress from one profile to the next.

Developments with their Patterns of Collaboration can also have an influence on their Skill Levels and Perceived Personal Relevance. If a student gets involved with a good study group or personal network that works collaboratively and has a positive programming learning experience, this may improve not only their Skill Levels but also potentially improve their Perceived Personal Relevance towards programming.

On the other hand, if a student goes through a negative learning experience of programming through difficulties in developing their problem solving behaviours, or frustrations in not being able to find information or obtain help from their personal networks, that may in turn have a negative influence on their Perceived Personal Relevance for programming.
6.4.3 Profile Preferences for Collaboration

This section presents and discusses a table describing the collaboration preferences of each Programming Learner Profile, highlighting the influences that the profiles have on Patterns of Collaboration and Personal Networks. Table 6-2 shows each Learner Profile and lists the Personal Networks they reported using, in order of preference, and also their dominant Patterns of Collaboration. This table brings together and describes the relationships between Programming Learner Profiles, Patterns of Collaboration and Personal Networks. It gives another perspective to better understand the many complex relationships.

The Reluctant Beginners are highly dependent and interact primarily with a One Way Dependency with teaching staff, help lab tutors, their personal networks within the degree and their wider personal networks. They predominately prefer face to face interaction because they have difficulty gaining understanding from written and online information sources.

The Willing Beginners predominantly interact with a Two Way Co-dependency interaction with their personal networks within their degree and wider networks, consulting teaching staff as a last resort. Keen Beginners consult their personal networks within their degree but work in a Collaborative Independent interaction with personal networks within their degree opting to consult teaching staff as a last resort.

The Budding Managers have learnt to leverage their personal networks to greatly enhance their learning. They actively interact with all their personal networks in a Co-Dependent fashion with a skill division of labour or task division of labour depending on the activity. They tend to become dependent for tasks that are complex and they are not perceived as relevant. In their preferences for personal networks they choose the networks that will be most beneficial to them.

The Budding Practitioners are more skilled and also work Collaborative Independently with their personal networks. Consulting personal networks including teaching staff is left as a last resort for advanced issues. This is also the case with Budding Developers and Advanced Developers with the difference that Advanced Developers usually do not have peers who are at their same skill level within their degree so if they need a pointer or direction, their only resort is teaching staff.
<table>
<thead>
<tr>
<th>Profile</th>
<th>Personal Networks in order of preference</th>
<th>Pattern of Collaboration</th>
</tr>
</thead>
</table>
| Reluctant Beginner      | • Class tutors  
• Help lab tutors  
• Personal networks within the same degree  
• Wider personal networks | • Highly Dependent  
• Prefer face to face interaction                                                                 |
| Willing Beginner        | • Personal networks within the same degree  
• Wider personal networks  
• Class tutors  
• Help lab tutors | • Co-Dependent  
• Task or skill division of labour  
• Consult teaching staff as last option                                                                |
| Keen Beginner           | • Personal networks within the same degree  
• Class tutors | • Collaborative Independent  
• Work independently  
• Collaborate with personal networks in their class  
• Provide help and/or compare approaches  
• Consult class tutors as a last option                                                                |
| Budding Manager         | • Personal networks within the same degree  
• Help lab tutors  
• Wider personal networks  
• Class tutors | • Co-Dependent  
• Skill division of labour for planning / debugging  
• Task division of labour for coding.  
• Prefer face-to-face interaction with complex issues or those not perceived as relevant |
| Budding Practitioner    | • Personal networks within the same degree  
• Wider personal networks  
• Class tutors | • Collaborative or Solitary Independent  
• Work independently  
• Collaborate with personal networks in their class  
• Seek help and/or compare approaches  
• Consult class tutors for advanced problems                                                                |
| Budding Developer       | • Personal networks within the same degree  
• Wider personal networks  
• Class tutors | • Collaborative or Solitary Independent  
• Work independently  
• Collaborate with personal networks in their class  
• Provide help and/or compare approaches  
• Seek direction from wider network  
• Consult class tutors for advanced problems                                                                |
| Advanced Developer      | • Personal networks within the same degree  
• Wider personal networks  
• Class tutors | • Collaborative or Solitary Independent  
• Work independently  
• Consult others is last option  
• Collaborate with personal networks in their class  
• Provide help and/or compare approaches  
• Seek direction from wider network  
• Consult class tutors for advanced problems                                                                |

Table 6-2 - Profile Preferences for Collaboration
6.4.4 Profile Information Behaviour

This section presents and discusses a table detailing the information behaviour preferences of each Programming Learner Profile, highlighting the influences that the profiles have on Patterns of Information Use and Information Sources. Table 6-3 shows each learner profile and lists the information sources they reported using, in order of preference, and also their dominant Patterns of Information Use. This table brings together and describes the relationships between Programming Learner Profiles, Patterns of Information Use and Information Sources. It gives yet another perspective to better understand the many complex relationships involved.

A strong Ownership of Learning and strong Learning Task Intent incline students towards seeking a deep understanding and practicing thorough and systematic Problem Solving Behaviours, which in turn influences a more evolved Pattern of Information Use. Keen Beginners to some extent practice Code Adaptation independently, but the more highly competent profiles such as the Budding Practitioners, Budding Developers and Advanced Developers are able to interpret and apply the information they find to solve coding problems. They also practice Code Adaptation and Comparative Analysis. Advanced Developers use their information sources to go Beyond the Scope of their studies.

On the other hand, a weak Ownership of Learning and weak Learning Task Intent incline students towards getting the task done, ad-hoc and unsystematic Problem Solving Behaviours, which in turn influences a basic Pattern of Information Use. Reluctant Beginners use information sources for simple syntax verification and finding the meaning of errors but are unable to relate the information they find to solve a coding problem. They have a tendency to look for complete answers as they need extensive guidance to do Code Adaptation. Willing Beginners and Budding Managers display a similar Pattern of Information use but are more able to perform minimal Code Adaptation and also are able to online forums more readily.

Reluctant Beginners are particularly contrasted in this table. Whereas most other profiles predominately prefer online sources as their first preference of information sources, the Reluctant Beginners prefer to seek face to face help from teaching staff or their personal networks, because they have difficulty understanding the other information sources available to them.
<table>
<thead>
<tr>
<th>Profile</th>
<th>Information Source in preference order</th>
<th>Patterns of Information Use</th>
</tr>
</thead>
</table>
| Reluctant Beginner          | • Face to face interaction with personal networks (help lab tutors, tutors, friends)  
• Skimming of written notes  
• Sometimes skip revision of tutorials  
• Google                                                                 | • Syntax Verification - notes and Google  
• Finding meaning of errors - Google  
• Unable to make use of information they find  
• Look for complete sample answers  
• Minimal code adaptation                                                          |
| Willing Beginner            | • Active use of Google  
• Online manuals and online tutorials  
• Limited external forums  
• Skimming of written notes                                                                 | • Syntax Verification - mostly Google  
• Finding meaning of errors - Google  
• Difficulty making use of information found  
• Look for complete or partial sample answers  
• Code adaptation with guidance                                                             |
| Keen Beginners              | • Very active use of Google  
• Online manuals and online tutorials  
• External forums and internal forums  
• YouTube videos  
• Revision of written notes                                                                 | • Syntax Verification - mostly Google  
• Finding meaning of errors - Google  
• Able to use information found to solve prob  
• Look for sample code with specific pointers  
• Code adaptation done independently                                                          |
| Budding Managers            | • Active use of Google  
• Online manuals and online tutorials  
• External forums and internal forums  
• YouTube videos  
• Revision of written notes                                                                 | • Syntax Verification - mostly Google  
• Finding meaning of errors - Google  
• Difficulty making use of information found  
• Look for complete or partial sample answers  
• Code adaptation with guidance, especially for more complex tasks                        |
| Budding Practitioners       | • Very active use of Google  
• Prominent use of external forums  
• Online manuals and online tutorials  
• YouTube videos  
• Attentive revision of written notes                                                      | • Syntax Verification - mostly Google  
• Finding meaning of errors – Google and External Forums  
• Able to use information found to solve prob  
• Look for relevant sample code with pointers  
• Code adaptation done independently  
• Comparative Analysis                                                                |
| Budding Developers          | • Very active use of Google  
• Prominent use of external forums  
• Read internal forums  
• Online manuals and online tutorials  
• YouTube videos  
• Attentive revision of written notes                                                      | • Syntax Verification - mostly Google  
• Finding meaning of errors – Google and External Forums  
• Able to use information found to solve prob  
• Look for relevant sample code with pointers  
• Code adaptation done independently  
• Comparative Analysis  
• Beyond the scope                                                                     |
| Advanced Developers         | • Very active use of Google  
• Prominent use of external forums  
• Use of internal Forums  
• Online manuals and online tutorials  
• YouTube videos  
• Attentive revision of written notes                                                      | • Syntax Verification - mostly Google  
• Finding meaning of errors – Google and External Forums  
• Able to use information found to solve prob  
• Look for relevant sample code with pointers  
• Code adaptation done independently  
• Comparative Analysis  
• Beyond the scope                                                                     |

Table 6-3 - Profile Information Behaviour
6.4.5 Summary

The Programming Learner Profiles themselves bring together the many influences that arose during the analysis, in a unified and cohesive manner to better understand them. There is a bi-directional influential relationship between Learner Nature and Learning Behaviours. The stronger relationship is the influence that Learner Nature has on Learning Behaviours, with Perceived Personal Relevance being the strongest influence.

Perceived Personal Relevance is highly influential on Ownership of Learning, Learning Task Intent and Problem Solving Behaviours. Students who perceive that learning programming is relevant to their future described having a deep desire to thoroughly understand their learning tasks and deliverables. On the other hand, the students who do not perceive programming as relevant to them are more inclined towards completing the course deliverables rather than learning the concepts.

Perceived Personal Relevance also influences Patterns of Collaboration and Information Use. Students with a higher Perceived Personal Relevance tend to interact with minimal reliance on others. They also progress towards thorough Problem Solving Behaviours more rapidly and the more evolved solution discoveries of code adaptation, comparative analysis and beyond the scope. On the other hand, students who have a lower Perceived Personal Relevance interact with higher dependence on others, from which they tend to seek complete answers which they may not understand. Their Problem Solving Behaviours tend to also be weak.

Learning Trait influences Ownership of Learning and also Problem Solving Behaviours in that students who tend to have an independent attitude, have a stronger ownership of their learning. Skill Level influences Learning Behaviours as a whole by influencing the concepts that make up Learning Behaviours - Patterns of Collaboration, Core Learning Perspectives and Patterns of Information Use.

The concepts that define Core Learning Perspective are closely related and intertwined. In general, students who have a strong Ownership of Learning also have a strong Learning Task Intent and practice thorough Problem Solving Behaviours. Similarly, students with a weak Ownership of Learning tend to also have a weak Learning Task Intent and Problem Solving Behaviours.

A stronger Ownership of Learning, Learning Task Intent and thoughtful and thorough Problem Solving Behaviours will influence a Pattern of Collaboration that is collaborative independent or solitary independent. Such students will also have a Pattern of Information Use that is more evolved, being able to identify, adapt and evolve code through Comparative Analysis and go Beyond the Scope.
There is also a feedback influence that occurs between Learning Behaviours and Learner Nature. As students develop and improve their Learning Behaviours, Patterns of Information Use and Problem Solving Behaviours, these will naturally improve their Skill Level. Developments with their Patterns of Collaboration can also have an influence on their Skill Levels and Perceived Personal Relevance, such as if a student gets involved with a good study group and has a positive programming learning experience, this may improve their Perceived Personal Relevance.

6.5 Chapter Summary

This chapter described the core of the model, Programming Learner Profiles, and also discussed the relationships between the various elements of the model. Programming Learner Profiles describe the varying patterns of behaviour in the way students approach the learning of programming and the way they use their information sources and personal networks.

The Programming Learner Profiles are made up of two core parts: Learner Nature (Perceived Personal Relevance, Learning Trait and Skill Level) and Student Learning Behaviour (Core Learning Perspective, Patterns of Collaboration and Patterns of Information Use).

Seven distinct Programming Learner Profiles were described: Reluctant Beginner, Willing Beginner, Keen Beginner, Budding Manager, Budding Practitioner, Budding Developer and Advanced Developer. Each profile describes, in essence, the nature of a different type of student.

Pathways that students tend to follow in relation to moving from one profile to another were also described. A number of relationships were detailed: how Learner Nature and Learning Behaviours (particularly Perceived Personal Relevance) influence each other, the interrelationships within the Core Learning Perspectives, and how Core Learning Perspectives influences Patterns of Collaboration and Patterns of Information Use.

This chapter completes the description of the theory of Influences on the Student Learning Experience of Programming. The next chapter enfolds the theory with the literature, as described in the research plan in chapter 3. Chapter 7 will discuss how the theory confirms and contrasts with previous studies and the contributions made by this study.
Chapter 7  Literature Comparison and Discussion

The Theory of Influences on the Student Learning Experience of Programming was presented in chapters 4, 5 and 6. This chapter enfolds this theory with the existing literature, presented in chapter 2. This is done as part of the research plan presented in chapter 3. Enfolding the theory with the literature is a key part of the Grounded Theory methodology.

This chapter will discuss how the theory confirms and contrasts with findings from previous studies. The discussion will also highlight the gaps in the literature and the contributions made by this study to the body of knowledge. This chapter is structured following the components of the theory:

- Student Characteristics – General
- Student Characteristics – Programming Context
- Personal Networks
- Information Sources
- Core Learning Perspective
- Patterns of Information Use
- Patterns of Collaboration
- Perceived Personal Relevance
- Learning Nature
- Programming Learner Profiles
- Seven Learner Profiles

7.1 Student Characteristics – General

As described in section 4.2, the characteristics of the students themselves would naturally have an influence on their learning. The general characteristics were those that related to the students themselves: age, gender, stage (year) of degree, degree enrolled and working commitments outside of study.
Age

In this study, each age group was represented in each of the profiles, so no clear patterns can be observed. This is largely in agreement with previous studies that found that age was not a good predictor of success in programming (Beise et al. 2003; Bergin & Reilly 2005b; Rountree, N et al. 2004; Woszczynski, Haddad & Zgambo 2005a, 2005b) (see section 2.1.5).

One trend to note from this study was that maturity in age may sharpen the student’s Perceived Personal Relevance, making the mature age students more likely to be at one extreme or the other.

Gender

Although there were somewhat less females than males in the study (22 males and 9 females), this was representative of the gender ratio of students in the Information Systems degree that was the primary source of participants.

Females in the study made up 4 of the 5 Reluctant Beginners and only 1 Advanced Developer. This is in contrast to previous studies that have found no correlation between gender and programming outcomes (Beise et al. 2003; Bennedsen & Caspersen 2005; Byrne & Lyons 2001; Chinn et al. 2010; Pillay & Jugoo 2005; Rountree, N, Rountree & Robins 2002; Ventura & Ramamurthy 2004) (see section 2.1.5). However, this could also be seen to confirm other previous studies that have found female students are less confident with their skills (Carter, J & Jenkins 2001). Females were also over-represented in the Budding Managers profiles, which may confirm studies that found that females tend to work more with others (Chinn et al. 2010; Sinclair & Kalvala 2015).

As the population is small, however, no definitive conclusions can be drawn either way.

Degree Enrolled

Previous studies into degree majors have had mixed results (see section 2.1.4). The intended major of Computer Science students has not been found to be a significant factor in programming learning outcomes (Bennedsen & Caspersen 2005; Rountree, N, Rountree & Robins 2002), but at the same time Computer Science majors have been found to have a higher probability of passing the first programming course than Information Systems majors (Beise et al. 2003).
In this study, the majority of the students were enrolled in Information Systems, so a comparison cannot be made with Computer Science students. One interesting finding was that the majority of the students from non-IS degrees (Accounting, Marketing and Business) were Keen Beginners and Advanced Developers. This may be because these students chose to learn programming as an elective and saw it as relevant to their future. Carter, J and Jenkins (2001) also found that students who took computing courses that were compulsory had a less positive experience than those who chose computing freely.

**Stage in Degree**

Previous studies have found no correlation between the year level of students and the results of introductory programming courses (Bennedsen & Caspersen 2005; Rountree, N, Rountree & Robins 2002) (see section 2.1.4). In general, students improve their understanding of programming concepts by the end of their second course (Gomes & Mendes 2008; Sheard et al. 2009; Tew, McCracken & Guzdial 2005).

The findings of this study overall agree with these previous studies, in that the students that have advanced to the higher Programming Profiles are in the later years of their degree. One exception, however, was the finding that some students remain as Reluctant Beginners, even into their final year after having passed several programming units.

**Working Commitments Outside of Studies**

No previous studies have been found that specifically investigate the impact that student’s working commitments have on their programming learning outcomes. It is common, however, for full-time students to have part-time jobs and part-time students are often working full-time.

The most interesting finding from this study in relation to employment was that while half the students worked while studying, half of the Budding and Advanced Developers, who were also full time students, reported also working between 20 and 29 hours per week. Some of these were also working on their own programming projects on the side. Those students that are keenly interested in programming seek out further opportunities to expand their skills.
7.2 Student Characteristics - Programming Context

As described in section 4.3, programming context student characteristics describe what the students bring to the learning situation in terms of their preconceived expectations, previous programming studies, prior knowledge and skills, mindset and self-efficacy.

Preconceived Expectations

In this study, almost half of the programming students expected to learn something useful, a quarter expected to learn it in less depth than they actually did, 15% expected programming to be difficult and 20% had no expectations.

A number of recent studies have explored student expectations of programming in relation to the computing discipline and computing careers. CS major students are more likely to regard programming as a major element of computing careers (Kinnunen et al. 2016), whereas non-CS/IT majors do not see programming as important to a career in computing (Courte & Bishop-Clark 2007; Uzoka et al. 2013).

This study confirms these findings, in that the students that expected programming to be useful were Keen Beginners, Budding Developers and Advanced Developers, who could be identified as similar to IT majors. The students that expected to learn less programming than they did are similar to the non-major students in the previous studies that do not see programming as important.

Previous Studies of Programming

Previous studies have shown that students with previous programming experience will tend to perform better in learning programming than students with no previous experience (Cantwell-Wilson & Shrock 2001; Chinn et al. 2010; Gomes & Mendes 2008; Hagan & Markham 2000; Holden & Weeden 2003, 2004, 2006; Kori et al. 2016) (see section 2.1.2).

This study also found that previous knowledge of programming also influences student performance, but the influence can be both positive and negative. The quality of a student’s previous studies with programming influences their Perceived Personal Relevance and learning experience. Those with a positive previous experience are more likely to have a higher Perceived Personal Relevance and reach the more competent profiles. Those with a previous negative experience, on the other hand, are more likely to stay reluctant and not be interested in learning programming.

These findings shed more light on the influence of previous programming knowledge.
Previous IT Knowledge or Skills

This study found that previous IT knowledge does, in most cases, influence the learning of programming (see section 4.3.3). All the students that were in the higher competent profiles (Budding Practitioner, Budding Developer and Advanced Developer) had studied IT previously, whereas most of the Reluctant Beginners only had basic previous IT skills.

These findings are similar to those of previous studies that have found students with previous programming experience tend to perform better than students without programming experience (see section 2.1.2).

Mindset

Mindset Theory has been developed by Dwek (2008) and describes how a person’s belief about their ability influences their response towards a goal (Cutts et al. 2010; Diener & Dweck 1980; Dweck 2000; Elliott & Dweck 1988; Robins, RW & Pals 2002). There are two mindsets: Fixed Mindset (learnt helplessness) and Growth Mindset (mastery orientation) (see section 2.2.5).

The majority of the students in this study showing a fixed mindset were from the Reluctant and Willing Beginner profiles, while the majority of students with a growth mindset were from the Keen Beginners and more competent student profiles (see section 4.3.4).

The interesting observation regarding Mindset in this study was that some students actually have mixed or multiple mindsets. Some of the students that display a fixed mindset towards programming actually display a growth mindset towards other areas such as multimedia, databases and networking. This is somewhat in line with the finding by Scott and Ghinea (2014) that mindsets may be domain specific, although their study found different mindsets between programming and intelligence in general. This study makes a contribution to this developing concept that students may have multiple mindsets, depending on the domain involved. In this study, Perceived Personal Relevance was found to be a major influence on the type of mindset students approached their programming studies with.
Self-Efficacy

Previous studies have found a positive correlation between a student’s self-efficacy of programming and their performance in programming courses (Adair & Jaeger 2011; Altun & Mazman 2015; Bergin & Reilly 2005a, 2005b, 2006; Gomes, Santos & Mendes 2012; Kanaparan 2016; Özmen & Altun 2014). Students who feel capable of writing programs tend to perform well, while students who feel they are not capable of writing programs perform poorly (see section 2.2.4).

The findings in this study were similar to those of the previous studies (see section 4.3.5). All of the Reluctant Beginners and two thirds of Willing Beginners described having a low self-efficacy with programming. Similarly, all of the more competent profiles had a high self-efficacy. Most of the students with a high self-efficacy also displayed a growth mindset towards programming. Similarly, the students with a low self-efficacy displayed a fixed mindset.

The low self-efficacy however, does not necessarily extend to other IT related areas. Five of the Reluctant Beginners (all females) with a low self-efficacy in programming described having a high self-efficacy with databases and HTML. This connects with the observation that students can have a mixed or multiple mindsets.

7.3 Personal Networks

Personal Networks in this study include a wide range of personal contacts beyond just the peers within the same class or year level. These Personal Networks expand to include friends in the same degree, friends in different degrees or universities, friends in industry and family members. Personal networks also include lecturers, class tutors and help lab tutors (students from the same degree but later stage).

Few previous studies have investigated the personal networks of higher education students in general (see section 2.5.3). One study by Tsai (2012) found that when higher education students have course related problems, they are most likely to consult peers, teaching assistants, professors and occasionally parents. When seeking moral support, they are most likely to consult their family members, friends, peers and occasionally teaching staff (Tsai 2010a, 2010b; Tsai & Kim 2013). More specific to programming, one study by Sheard et al. (2013) found that few programming students sought help from teaching staff, with those who did usually consulting a help desk tutor but most students consulted their friends online.
The findings in this study are similar to the findings of these previous studies but this study has gone into more depth in the way programming students use their personal networks and it also describes how the different student Programming Profiles interact with their personal networks differently.

In this study, the bulk of student interactions with their personal networks were found to happen outside the classroom. Sometimes students consult multiple personal networks for different levels of help and assistance. Informal interactions begin in the classroom but then evolve into more regular interactions outside the classroom. Many of these regular consultations take place face to face or via online media, such as text messaging, Facebook or email. In contrast to studies that found students use Facebook only for socialising (Cheung, Chiu & Lee 2011; Tian et al. 2011), this study found that Facebook is indeed used by students to communicate with their personal networks for educational purposes, although not necessarily using educator provided Facebook groups.

When seeking help from teaching staff, students in this study choose the most convenient source, depending on the type of issue, urgency and time available. Students who lack confidence may feel intimidated to consult the lecturer because of fear of judgement – a similar finding to that of Shi (2016). Most novice students prefer face to face consultations, as they require a richer explanation. More confident students consult teaching staff for advanced issues after exhausting all other avenues.

Half of the students in this study consulted the help lab, which are staffed by fellow students in later years. The help lab mentors tend to be from the more advanced Programming Profiles in this study and gain benefit from helping students, a similar finding to D'Souza et al. (2008). One new finding in this study is that in some cases, the help lab tutors become part of the students' wider personal network and are consulted for help outside of the help labs.

7.4 Information Sources

Students in this study used a very wide variety of information sources. Written sources included subject materials, notes and books. Online sources included: Google, online manuals, online tutorials, YouTube, external programming forums and internal course forums.
Relatively few previous studies have investigated the range of information sources that programming students use (see section 2.5.3). Studies by Costa, Aparicio and Pierce (2009) and Blaho et al. (2013) found that students preferred online sources. The types of sources they reported being used were: online help, product help, books, lecture notes, forums and peers. Sheard et al. (2013) also reported that students used the Internet first when looking for information.

The findings of this study largely support the findings of these previous studies, but this study has gone into more depth in the way the students make use of each information source and also describes how different student programming profiles use the same information sources in different ways. For example, the beginners only use Google to check syntax because they cannot understand any sample code they might find, whereas advanced students use Google to investigate advanced concepts.

Internal forums are online communities provided to the students (see section 4.5.6). In this study, the Beginner profiles found interpreting the help that is posted difficult to follow, but they were relieved knowing that other students faced similar problems. Some were reluctant to post for fear of judgement, mainly from other students. The students who did post tended to be the more advanced students, but they did tend to consult external forums first.

These findings largely support those of previous studies described in section 2.5.4. Shaw (2012) reported that students that are ‘repliers’ have higher grades, which corresponds to findings in this study that the more advanced programming profiles are the ones who answer posts. Shi (2016) found students searched for information online to avoid asking “silly” questions and being judged by their lecturers and peers.

Those studies that have trialled using Facebook as internal forums have found students enthusiastic to use them (Hundhausen & Carter 2014; Maleko et al. 2014; Maleko et al. 2013; Pieterse & Rooyen 2011). While the students in this study were not provided Facebook groups by their lecturers, they did, however, report using Facebook to communicate with their student personal networks and discuss programming problems and exchange code privately (see section 4.4.1).

In this study, half the students used external forums such as Stack Overflow, but only a third of those posted questions (see section 4.5.5). Students, especially part-time, found these a convenient information source, especially outside classroom hours. They were used as a quick reference, as a learning resource and as a guide for how to code advanced features. Students place a high level of trust in these forums, based on the belief that thousands of users would have faced similar problems.
These findings are complementary to those of other studies into public programming forums (see section 2.5.5). As in this study, other studies have also found that most students tend to only read external forums and not post (Hsiao 2015a, 2015b; Hsiao & Naveed 2015). Lu and Hsiao (2016) found that novice students only skimmed through posts and had difficulty formulating queries, whereas advanced students were able to refine their queries. Similarly, this study found that the novice students had difficulty understanding code they might find online whereas the advanced students were better at searching for and understanding code found on external forums.

This study contributes to the understanding of how programming students use information sources by going into more depth in the way different programming profiles use the same information sources in different ways. Another contribution is the finding that even though internal forums are provided to students, beginners have trouble interpreting and applying information in the posts to solve problems. Also, advanced students often consult external programming forums before looking at the internal forum.

7.5 Core Learning Perspective

Core Learning Perspectives describes the key attitudes, intentions and behaviours of students as they proceed with their learning of programming. Core Learning Perspectives brings together three concepts: Ownership of Learning, Learning Task Intent and Problem Solving Behaviours. Ownership of Learning, Learning Task Intent and Problem Solving Behaviours are closely related and intertwined.

As described in the following sections, Ownership of Learning and Learning Task Intent are conceptually similar to the previously researched concepts of Deep and Surface learning and Intrinsic and Extrinsic motivations. Ownership of Learning and Learning Task Intent, however, provide deeper insights into these previously understood concepts. Ownership of Learning represents the attitude of the students towards their learning, while Learning Task Intent represents their intention to their actions.

The Programming Learning Profiles also provide a deeper insight into how Ownership of Learning and Learning Task Intent influence Learning Behaviours (Problem Solving Behaviours, Patterns of Collaboration and Patterns of Information Use). For example, Reluctant Beginners have a weak Ownership of Learning, a weak Learning Task Intent and Ad-hoc Problem Solving Behaviours (lack of planning, ad-hoc trial and error, unsystematic approach, dependent debugging and coding in big chunks). They also tend to have one-way dependency on others and use simple Patterns of Information Use (syntax verification and looking for sample answers).
### 7.5.1 Ownership of Learning

Ownership of Learning describes the students’ view on who is responsible for their learning – whether they take responsibility for their learning or assume others are responsible for teaching them. Students with a Strong Ownership of Learning want to work out their problems independently. They consult multiple sources, do additional practice outside the classroom and work collaboratively with others. Students with a Weak Ownership of Learning expect to be taught. They take short cuts and interact dependently with others. Students with an intermediate Ownership of Learning have mixed views about their learning responsibility, depending on which topics they see as relevant to them.

The concept of Ownership of Learning in this study is very similar to a number of concepts already found in the literature: Deep and Surface learning, Intrinsic and Extrinsic motivations and Dependent and Independent learning (see sections 2.3.1, 2.2.1 and 2.2.3). Students with a strong Ownership of Learning exhibit similar characteristics to those with a Deep Approach to learning, a weak Ownership of Learning is similar to a Surface Approach to learning and an Intermediate Ownership of Learning is similar to an Achievement Approach to learning (Biggs 1989; Biggs & Tang 2007, 2011; Carbone 2007; Cope & Prosser 2005).

Similarly, Ownership of Learning is conceptually similar to Intrinsic and Extrinsic motivation. Comparing to Ryan and Deci (2000), a strong Ownership of Learning maps to Integrated and Intrinsic motivations, a weak Ownership of Learning maps to Amotivation and External Regulation, while an Intermediate Ownership of Learning maps to Introjection and Identification (Ryan & Deci 2000). The concept of dependence as described by Baird and Mitchell (1991) is very similar to a weak Ownership of Learning. The behaviours of students with a strong Ownership of Learning are similar to the strategies used by students that have a high level of Self-Regulated Learning (Micari & Light 2009; Pintrich 1999).

Conley and French (2014) also use the term ‘Ownership of Learning’ to describe their model of student readiness for college. The components of their model (motivation and engagement, goal orientation and self-direction, self-efficacy and self-confidence, metacognition and self-monitoring and persistence) are similar to the characteristics described for students with a strong Ownership of Learning in this study.
7.5.2 Learning Task Intent

Learning Task Intent describes the students’ intentions towards their tasks - whether they want to understand them or just get them done. A strong Learning Task Intent describes the intention to gain a deep understanding whereas a weak Learning Task Intent describes the intention to complete the tasks in order to get the task done. An intermediate Learning Task Intent describes the intention to understand only those parts of the task perceived as relevant to them.

The concept of Learning Task Intent in this study is very similar to some of the elements found in Deep and Surface learning and Intrinsic and Extrinsic motivations (see sections 2.3.1 and 2.2.1). Biggs and Tang (2011) describe a Surface Approach to learning as arising from an intention to get the task out of the way with a minimum of effort, which usually involves cutting corners. This is very similar to a weak Learning Task Intent. A Deep Approach to learning, on the other hand, involves students using strategies to maximise their understanding and to satisfy their curiosity. This is also very similar to a strong Learning Task Intent (Biggs & Tang 2011). Similarly, Learning Task Intent is conceptually similar to Intrinsic and Extrinsic motivation (Ryan & Deci 2000).

7.5.3 Problem Solving Behaviours

This study found a wide range of Problem Solving Behaviours described by the students that have been grouped as Thorough, Intermediate and Ad-hoc (see section 5.1.3). Thorough Problem Solving Behaviours include: thoughtful planning, thoughtful error detecting, desk checking and efficient and independent debugging. Intermediate Problem Solving Behaviours include: simple planning, thoughtful trial and error, coding step by step, developing independent debugging skills and relying on sample code.

Students who display Ad-hoc Problem Solving Behaviours tend to randomly practice or skip some phases in the programming lifecycle. They skip planning, make ad-hoc changes to code, have an unsystematic approach, practice dependent debugging by repeatedly seeking help from others and try to code in big chunks. They begin their assignments with misunderstood or incomplete learning tasks.
Most of these problem solving behaviours have been previously documented and explored in the literature (see section 2.4). In studies looking at the use of strategies (see section 2.4.1), novices have been found to dive into coding without a clear understanding of the problem, which is the same as the Ad-hoc Unsystematic Approach in this study (Carbone 2007; Carbone et al. 2009). Perkins et. al. (1989) identified Movers that systematically work though a problem, which is similar to Thoughtful Error Detecting and Thoughtful Trial and Error in this study. Tinkerers, on the other hand, tinker with their code in the same way as described by Ad-hoc Trial and Error in this study (Carbone et al. 2009; Perkins et al. 1989). Further examples of Ad-hoc Trial and Error have been described by Murphy et al. (2008).

McCartney et al. (2007) described strategies used by successful students, many of which correlate with the intermediate and thorough problem solving behaviours in this study. Sheard et al. (2013) described students relying on a model solution, which is the same as Relying on Sample Code in this study.

Previous studies that looked at planning strategies (see section 2.4.2) found that students have difficulties in planning and putting the pieces together (de Raadt 2007; Soloway 1986). Experts rely on their tacit knowledge built up from solving past problems (Soloway 1986). This study found similar behaviours of students. Most of the students had difficulties planning out the program before starting, either practicing very simple planning or no planning at all. The students that practiced Thoughtful Planning were similar to the ‘experts’ described by Soloway (1986) as they were mostly from the Budding and Advanced Developer profiles and were using their tacit knowledge to draft plans for new tasks.

Studies into code comprehension (see section 2.4.3) found that novices tend to have difficulties reading code and comprehending what the program does (de Raadt 2007; Lister et al. 2004; Robins, A, Rountree & Rountree 2003; Rountree, J et al. 2005). Even good students have had difficulties understanding other people’s written code (Ahmadzadeh, Elliman & Higgins 2005) or even their own code while attempting to change it (Özmen & Altun 2014). Similarly, in this study, the Beginner profiles described having difficulty comprehending code they found online. Students described searching for sample code that was close to a complete answer to their problem (section 5.3.3), a behaviour also described by Sheard et al. (2013). In contrast, the Budding and Advanced profiles described practicing more evolved solution discoveries such as Code Adaption, Comparative Analysis and Beyond the Scope which require the ability to read and understand other people’s written code as well as their own code (see sections 5.3.4, 5.3.5 and 5.3.6).
Previous studies into fragile knowledge (see section 2.4.4) found that fragile knowledge, defined by Perkins and Martin (1986) as the possession of and ability to articulate knowledge, but inability to apply it to write a program, may explain why novices have difficulties problem solving programming problems (de Raadt 2007; de Raadt, Watson & Toleman 2006; Lister et al. 2004; Robins, A, Rountree & Rountree 2003). In this study, Ad-hoc Trial and Error and using an Unsystematic Approach are examples of students having fragile knowledge.

Previous studies into tracing of code (see section 2.4.5) have found that the ability to trace code is essential for program comprehension (Robins, A, Rountree & Rountree 2003). Novices have difficulties with close tracking (Perkins et al. 1989) and require assistance to track and isolate code problems (Carbone et al. 2009). This study has found similar findings in that most of the Reluctant Beginners, Willing Beginners, and Budding Managers practice Dependant Debugging as they find code tracing difficult. The more Advanced Profiles practice Thoughtful Error Detecting which requires proficiency in code tracing. Students that are Developing Independent Debugging use similar techniques to those described by Murphy et al. (2008) such as tracing mentally and using print statements. Efficient and Independent Debugging matches the more effective debugging methods described by Murphy et al. (2008): the advanced profiles described using the debugging tool to trace code.

Previous studies into code writing difficulties (see section 2.4.6) have identified that novice programmers have difficulty combining individual statements into valid programs (Winslow 1996). Özmen and Altun (2014) found that students avoid practicing programming, which constrains them from remembering syntax (Özmen & Altun 2014). This study found similar findings to these previous studies, particularly with the Beginner profiles, that had difficulties writing code. The Ad-Hoc problem solving practice of Learning Task Skipping is similar to the programming practice avoidance behaviour found by Özmen and Altun (2014).

Studies on debugging strategies and difficulties (see section 2.4.7) identified debugging as one of the most difficult programming phases novices are exposed to (Murphy et al. 2008), that even high achieving students can struggle with (Rodrigo et al. 2013). Ineffective tinkering generates many compilation errors that students spend much time on (Carbone et al. 2009; Rodrigo et al. 2013).
This study found many similar debugging behaviours to those reported in previous studies, such as Ad-hoc Trial and Error and Coding in Big Chunks, which make debugging more difficult. The thorough problem solving behaviours: Thoughtful Error Detecting, Desk Checking and Efficient and Independent Debugging in this study are similar to the strategies described in the studies by Ducasse and Emde (1988), McCauley et al. (2008) and Murphy et al. (2008). The most interesting finding in this study in relation to debugging was the Dependent Debugging practiced by some Beginner profiles, which involves the student relying on someone else to find the bugs for them.

In summary, most of the problem solving behaviours identified in this study have been reported in previous studies. This study, however, presents these behaviours in a unified way that includes Ad-hoc, Intermediate and Thorough behaviours. This provides a new perspective on problem solving behaviours.

7.6 Patterns of Information Use

Patterns of Information Use describe the different ways students use written and online sources (see section 5.3). At a basic level, students look up their information sources for syntax verification and checking error messages. There are a number of methods of Solution Discovery. Looking for sample answers involves looking for a section of code that matches the students’ problem. Code Adaptation involves identifying subsections of code written by others that could be adapted. Comparative Analysis involves students comparing their code to others to improve their own coding style. Beyond the Scope involves using online sources to learn new languages or features not covered in the curriculum.

Previous studies into how programming students use their information sources have been rather limited (see section 2.5.3). Edwards (2005) identified four conceptions of information search, but those were for university students in general: Looking for a needle in a haystack, Finding a way through a maze, Using the tools as a filter and Panning for gold. The focus on Edward’s study was on how well students understand and use the search tools available to them. The Patterns of Information use is similar in this regard in that it shows how different types of students make use of the same information sources in different ways.
There has been considerable research into problem solving behaviours of programming students (see section 2.4). Some of the strategies investigated in these studies involve information use and can be somewhat compared to Patterns of Information Use. Novices have been found to copy from similar solutions (Winslow 1996). Novices also lack the knowledge and skills to effectively read and write code (Lister et al. 2004). McCartney et al. (2007) found that successful students learn from other people, tools or written materials.

Programming specific information seeking studies have mostly focussed on the frequency of usage of information sources, rather than the way the students use their information sources (Blaho et al. 2013; Costa, Aparicio & Pierce 2009). Shi (2016) explored information usage in more depth, finding that students preferred online sources for convenience, avoiding asking “silly” questions and for further investigation of concepts presented in lectures. Sheard et al. (2013) found that Google was the first port of call for students, a source of alternative explanations and a way to obtain sample code. They also found that novice students like to work from a model solution, whereas successful students can choose the level of granularity to look at the code. There are some similarities between these findings and the Patterns of Information Use.

This study, however, has gone into further depth than these previous studies in developing a programming specific set of information behaviours: Patterns of Information Use. These behaviours provide a new perspective on how programming students use information sources but also how the same sources are used differently by different types of students. These patterns are a major contribution of this study: Syntax Verification, Finding the Meaning of Errors Messages, Looking for Sample Answers, Code Adaptation, Comparative Analysis and Beyond the Scope.

7.7 Patterns of Collaboration

Patterns of Collaboration describe the different ways in which students interact with others (see section 5.2). The Model of Dependency (see Figure 5-1) highlights how much students rely on their personal networks. Although students interact with all or most of their personal networks, each Programming Learner Profile does so with a different level of dependency.
This study identified four distinct levels of dependency: One Way Dependency, Two Way Dependency, Collaborative Independent and Solitary Independent. In a One Way Dependency, the student seeking help is mostly or totally dependent on the peer, who acts as a crutch or as delegation service. In a Two Way Co-Dependency, students rely on or are co-dependent on each other (skill co-dependency or division of labour). Collaborative Independent students practice a two way interaction with peers that they trust. They compare each other’s approach to gain a better understanding. Solitary Independent students learn by helping others but they themselves do not seek help from their peers but may seek help from teaching staff for advanced problems. Students can interact with multiple personal networks concurrently, at different levels.

Dependence and independence has been previously studied (see section 2.2.3). Dependence has been described by Baird and Mitchell (1991) as a poor learning tendency and was highlighted as a deficit in generic skills by Carbone et al. (2009). Conley and French (2014) explore independence with their student ownership of learning model (motivation and engagement, goal orientation and self-direction, self-efficacy and self-confidence, metacognition and self-monitoring and persistence). Students who perform well in programming were found to use more meta-cognitive and resource management strategies than the lower performing students (Bergin, Reilly & Traynor 2005).

The concept of dependency described in this study is similar to those of previous studies. Previous studies have delved into the influences on dependence and independence but not the forms of interaction with personal networks. This study makes a major contribution to this field by giving a richer description of the different forms of dependency: One Way Dependency, Two Way Dependency, Collaborative Independent and Solitary Independent. Also, a deeper understanding of Patterns of Collaboration is provided by contrasting these forms of dependency with the Student Programming Profiles: e.g. the One Way Dependent students tend to be the Reluctant Beginners and the Solitary Independent students are the Budding and Advanced Developers.

Patterns of Collaboration and the Model of Dependency also provide some new insights into the area of collaborative learning (see section 2.5.1). Previous studies into student mentoring have explored various strategies that educators have used to enable students to mentor and help their peers (D'Souza et al. 2008; Devey & Carbone 2011). This study has explored the interactions between students and their peers as they occur outside the classroom and sheds new light on how students may help and mentor each other privately.
7.8 Perceived Personal Relevance (PPR)

Perceived Personal Relevance was a key concept in this study that emerged as having the largest influence on how much a student was motivated to learn programming. Students who were most interested in programming saw it as relevant to their future career goals and knew they wanted to work as programmers. On the other hand, students that perceived that they would never work as programmers had a very low level of interest in learning programming.

In the programming domain, career relevance has not been extensively explored. This may be because programming has traditionally been taught in Computer Science and it might be assumed that all CS students intend to work in the IT field (Kinnunen et al. 2016). A few studies, however, have found that not all programming students want to become programmers. Curzon and Rix (1998) found that programming was seen as a useful secondary skill but not a career. Chilana et al. (2015) identified students as “conversational programmers” - those who want to speak the language of programming to improve their job prospects. Some CS student plan to start their own company or work in small start-ups or as managers, rather than as programmers (Kinnunen et al. 2016).

No previous study, however, has investigated the influence of the student’s intended career goals on the learning of programming. This study makes a major contribution to the field by highlighting the potential impact of student’s career goals on their motivations to learn programming. How students perceive programming as being relevant to their future career goals arose as a key influence over many of the behaviours students exhibit in their learning. In some cases this single influence may outweigh all others.

It is highly influential on their Learning Behaviours: Core Learning Perspective, Patterns of Collaboration and Patterns of Information Use. For example, students that do not see programming as relevant to their future, are more likely to have a weak Ownership of Learning and focus on getting the task done rather than understanding. They will tend to practice ad-hoc problem solving behaviours. They are more likely to be dependent and reliant on others for their learning. They will only practice basic solution discovery patterns such as looking for sample answers.
7.9 Learner Nature

The Student Programming Profiles were defined by three dimensions grouped into Learner Nature: Perceived Personal Relevance, Learning Trait and Skill Level. Perceived Personal Relevance is discussed in the previous section and this section discusses Learning Trait and Skill Level.

7.9.1 Learning Trait

Learning Trait describes how dependent or reliant the students are on others (teachers, peers and their personal networks). Learning Trait has two values (dependent or independent) and is a summary of the Model of Dependence (see sections 6.1.2 and 5.2). A Dependent Learning Trait describes a passive approach to completing a task, heavily relying on others. An Independent LearningTrait describes a proactive approach, figuring out problems mostly or totally independently.

Dependent and independent learning has been previously studied, and Learning Trait is very similar in concept to these previous studies (see section 2.2.3). Dependence has been described by Baird and Mitchell (1991) as a poor learning tendency where the learner, despite being keen to succeed, adopts a passive, dependent approach to learning expecting to be told what to do and how to do it (Baird & Mitchell 1991; Carbone 2007). Micari and Light (2009) highlight three approaches to learning in peer-led workshops: reliance, engagement and independence. Sheard et al. (2013) found most programming students desired to work alone and understand by themselves, whereas others wanted to be taught.

Self-Regulated Learning (SRL) is defined as the degree to which learners are meta-cognitively, motivationally and behaviourally active participants in their own learning (Zimmerman 2002). The cognitive, metacognitive and resource management strategies represent the “skill”, whereas the motivational beliefs represent the “will” (Bergin & Reilly 2006; Pintrich 1999). Studies have found SRL to have a significant positive correlation with academic achievement (Bergin & Reilly 2006).

Learning Trait was one of the key defining concepts in the development of the Programming Learner Profiles. This highlights that certain types of student tend towards certain Learning Traits (see Table 4-1). The dependent students tend to be from the Reluctant Beginner and Budding Manager Programming Learner Profiles. The independent students are from the Willing Beginner, Keen Beginner, Budding Practitioner, Budding Developer and Advanced Developer Programming Learner Profiles. Although previous studies have identified these traits in programming students, this study sheds more light on to which types of students tend to be dependent or independent.
7.9.2 Skill Level

In this study, the programming skill level of the students was described as ranging from novice, to competent and advanced (see section 6.1.3). While many previous studies have sought to understand the learning experience of novice programming students, few studies have provided a definition of what constitutes a novice programmer as opposed to other skill levels. Dreyfus, Dreyfus and Athanasiou (2000) categorise the stages of developing from novice to expert in general: novice, advanced beginner, competence, proficiency and expert. Winslow (1996) provides a description of the difference between novices and expert programmers. The categorisation of three skill levels used in this study was developed for the purposes of this study, but is similar to those of the previous studies mentioned.

7.10 Programming Learner Profiles

The Programming Learner Profiles developed in this study are a form of learner categorisation of programming students. Each profile describes, in essence, the nature and behaviour of a different type of student. The profiles were defined by three dimensions that surfaced during the analysis and were grouped as Learner Nature: Perceived Personal Relevance, Learning Trait and Skill Level.

A number of previous studies have created programming learner categorisations, as discussed in section 2.6. Most of these studies involved interviewing first year students (Bruce et al. 2004; Dunican 2006; Jenkins & Davy 2000; Perkins et al. 1989), while Cardell-Oliver (2014) developed narrative categories from analysing learning artefacts and Robbins et al. (2003) based their categories on a literature review of other studies.

One major difference between the categorisations in this study and those of previous studies is that previous studies largely defined their categories along a single dimension, that of programming skill or ability. This study, however, used a combination of Perceived Personal Relevance, Learning Trait and Skill Level to define the Programming Learner Profiles. This has resulted in seven learner profiles that offer a richer picture of the interplay of key influences on programming learning behaviours.
Another major difference with previous categorisation studies is that most of these previous studies were focussed on first year and novice programming students. This study, however, sought to understand not only the experience of novice programming students but a more holistic understanding of the student experience throughout their degree and including post-graduation. Also, in contrast to these previous studies, this study sought to understand the student experience in the extended learning environment and not only in the classroom. The categories developed in this study therefore go beyond those of previous studies as they include students at all stages of their learning and explore their learning beyond the classroom.

Previous categorisation studies primarily described their categories in relation to their programming and problem solving behaviour. Whilst the Programming Learner Profiles in this study similarly describe student’s programming and problem solving patterns, this study goes beyond previous studies by also describing the information sources the different Programming Learner Profiles prefer to use, their Patterns of Information Use and the varying Patterns of Collaboration as applicable to each profile.

The categorisations in this study are therefore more holistic and comprehensive than previous programming learner categorisations.

7.11 Seven Learner Profiles

Seven distinct Programming Learner Profiles were developed during the analysis: Reluctant Beginner, Willing Beginner, Keen Beginner, Budding Manager, Budding Practitioner, Budding Developer and Advanced Developer (see section 6.2). Each profile describes the nature and behaviour of different types of students. This section compares and contrasts these profiles with other studies that developed learner categorisations (see section 2.6).

Challenging Nature of Programming (Dunican)

The Reluctant Beginners correspond to the Disengagers and Sloggers in Dunican’s study (Dunican 2006). Like Reluctant Beginners, Disengagers and Sloggers also find the material difficult to make sense of, seek help from their peers and usually look for a full solution. They also skip over the materials and have a negative self-efficacy towards programming. Whereas Disengagers give up after a short period of time, the sloggers persevere for longer.
One difference between Dunican’s Disengagers and Reluctant Beginners is that Dunican asserts that the students disengage and lose interest because they find the material too difficult to understand. This study found that Reluctant Beginners may be disengaged from the very beginning because their Perceived Personal Relevance is low at the start. Another difference is that this study found Reluctant Beginners are highly dependent on their personal networks, using a one-way dependency form of interaction. While Dunican alludes to Disengagers and Sloggers seeking help from their peers and staff, he does not elaborate this aspect of their interaction any further. Dunican also did not make any observation of how the students use online resources.

Willing Beginners correspond to both the Sloggers and Workers in Dunican’s Study (Dunican 2006). Those Willing Beginners who lose interest when concepts become hard are similar to the Sloggers. They skip tasks and move on without fully understanding. Theypersevere, looking at multiple sources repeatedly, use ad-hoc problem solving strategies and require assistance from others. The Willing Beginners who develop an interest and enjoyment of programming are similar to the Workers in Dunican’s Study.

One Difference with the Sloggers and Workers is that Willing Beginners tend to be Co-Dependent with their personal networks opting for a Skill or Task Division of Labour. They also prefer to seek coding examples online prior to consulting teaching staff and they practice basic patterns of information use. Dunican alludes to these behaviours but does not describe to the same depth the type of co-dependent interaction or the pattern of information used by his learner categories.

The Keen Beginners correspond to Dunican’s Manager and Achiever categories. They are similar to Dunican’s Manager and Achiever in the way they have a positive self-efficacy towards programming, their diligence, how they persevere through programming problems and in the more thoughtful problem solving behaviours they practice. They are also similar in that they engage in additional practice, re-implementing difficult code in order to build up their understanding prior to moving on to following topics. Managers engage with their peers in a reciprocal manner, similar to the way Keen Beginners interact with their personal networks in a collaborative independent fashion. Keen Beginners are also similar to the Achievers in the intrinsic interest they have with learning programming as well as the sense of accomplishment they feel when they complete challenging work.
Some differences found are that Keen Beginners see programming as highly relevant to their future careers and tend to go beyond of what is required in their courses. Dunican does not consider career goals in his category definitions. They display Patterns of Information Use that are intermediate in that they look for sample answers and are able to extract parts and/or implement some code adaption. They are also Collaborative Independent not only in the planning but in the debugging phase as well. Dunican does not describe these characteristics in the same depth.

As Dunican only considered first year students in his study, the remaining Programming Learner Profiles (Budding Manager, Budding Practitioner, Budding Developer and Advanced Developer) do not have direct correlations with Dunican’s learner categories, as these profiles are predominately of later year students.

Budding Managers do not see programming directly relevant to their career goals, which tend to be towards business analysis and IT management, preferring to solve problems at a higher abstract level. Budding Managers roughly correlate with Dunican’s Worker category in that they have a desire for achievement. They have mixed states of mind where they find programming hard and experience self-doubt but work through difficulties at a basic level in order to gain a manageable level of understanding until they encounter the next difficulty. They are also similar in the way they seek one-way assistance or work together with peers in a reciprocal fashion.

One major difference is that Budding Managers are good at leveraging their personal networks to achieve their learning goals, working dependently or co-dependently with others. They do this particularly with concepts or phases they do not see as relevant to them, such as debugging. Dunican does not emphasise leveraging personal networks in his study.

Budding Practitioners see programming as relevant to their future career goals, viewing it as a stepping stone to becoming better IT practitioners, but they do not intend to become programmers. Budding Practitioners share some similarities to Dunican’s Manager and Achiever categories, in the same way that Keen Beginners do. They differ, however, in that Budding Practitioners do seek a deep understanding of programming, but their focus is on the end product and the technologies used, rather than the programming itself. Their primary career path is to be an IT practitioner where knowledge of programming may be relevant but not a goal in itself. This characteristic is not mentioned in any of Dunican’s categories, so there are no categories in Dunican’s study that are similar to Budding Practitioners.
Both the Budding Developers and Advanced Developers see programming as highly relevant to their future career goals as they consider doing some form of programming work. Both the Budding Developers and Advanced Developers share some similarities to Dunican’s Achiever category, in the same way that Keen Beginners do, but they have a higher level of self-efficacy, approaching difficult tasks as things to be mastered.

One difference is that the Budding Developers and Advanced Developers in this study have a deeper curiosity to explore and go beyond what is required. Dunican mentions in his Achiever category that when the students do not find the material challenging, they do their own thing, but does not go into this in any more depth. They use external forums extensively, comparing and contrasting multiple sources of information to find better coding approaches. Dunican does not explore the use of forums in his study.

The Advanced Developers have reached the mindset of a professional developer, where their focus goes beyond the coding, considering issues of code architecture and flexibility. Also, both the Budding Developer and the Advanced Developer actively program in the real world while doing their studies. These aspects are not mentioned by Dunican, as his study only involved first year students.

Dunican described the Achiever category as being aware of other students’ struggles, but that this did not impact on their progression. The Advanced Developers and Budding Developers in this study are not only aware of the difficulties of other students but characteristically go further to help these students as they themselves learn by helping others. Budding Developers and Advanced Developers may also be solitary independents where they help others but do not seek help from their peers unless they believe they are on the same page as them.

Ways of Experiencing the Act of Learning to Program (Bruce Et Al)

Bruce et al. (2004) developed five categories of the ways student experience learning to program: Following, Coding, Understanding and Integrating, Problem Solving and Participating. Bruce’s categories differ from the Programming Learner Profiles in this study in that Bruce’s study was primarily focused on the outcome of the learning experience whereas this study looked more deeply at the influences behind the outcome. The Programming Learner Profiles have considerable overlaps with Bruce’s categories (Bruce et al. 2004).
Bruce’s Following category maps roughly to the Reluctant Beginners, Willing Beginners and Budding Managers. They are similar in that their motivation to learn programming is experienced as getting through the unit and focusing on the tasks that gain marks. They also experience frustration if their expectations are not met in terms of the material, its delivery or feedback from teaching staff. A major difference is that this study found the Reluctant Beginners to be highly dependent on their personal networks.

Bruce’s Coding category maps roughly to the Willing Beginners and Budding Managers. They are similar in that they seek guidance from others and rely on solutions provided to them by teaching staff. One main difference is that the Budding Managers leverage their social networks to improve their learning outcomes, which is not mentioned by Bruce et al.

Bruce’s Understanding and Integrating category roughly maps to the Keen Beginners and Budding Practitioners. They are similar in that they seek a deep understanding of the concepts: They experiment with the code, they refer to multiple sources of explanations and they use a building block approach to coding to improve their understanding. One main difference is that Keen Beginners and Budding Practitioners see programming relevant to their future careers. They work collaboratively independently with others and use external forums extensively with intermediate Patterns of Information Use.

Bruce’s Problem Solving category can be roughly mapped to the Budding Practitioners and Budding Developers in this study. They share the same similarities discussed between the Understanding and Integrating category and the Keen Beginners. One main difference is that Budding Practitioners prefer to solve problems with technology not necessarily with programming. Another difference is that Budding Developers display more thorough problem solving behaviours in all the programming lifecycle phases including debugging skills. Bruce only mentions coding and planning.

Bruce’s Participating Category maps to the Advanced Developers in this study. They are similar in that Advanced Developers think like programmers and share the same language and culture of the programming community. These students communicate with other programmers to learn the programming culture as described by Bruce. Advanced Developers practice these by participating extensively in external forums and they actively program in the real world.
There are some major differences between this study and that of Bruce et al. (2004). While Bruce’s study was based only on first year students, this study focused on students all throughout the degree and post-graduation. While Bruce mentions each category’s learning approach, this study gives a richer picture of the different learning approaches in the Core Learning Perspectives, which include Problem Solving Behaviours. As part of Learning Behaviours, this study also included Patterns of Collaboration with Personal Networks and Patterns of Information Use with Information Sources in greater depth than Bruce et al. Perceived Personal Relevance was also found to be a major motivation in this study, one which was not identified by Bruce et al. (2004).

**Stopers and Movers (Perkins Et Al)**

In the Perkins et al. (1989) study, the researchers conducted observations of novice programming students and asked them questions about the tasks as they were being attempted. Perkins’ study yielded three types of novices in terms of how they fix programming errors and mistakes: Stopers, Movers and Extreme Movers (tinkerers). The Problem Solving Behaviours concept in this study roughly correlates to Perkins’ types of novices with the emphasis being more on the influences behind the behaviours.

As Perkin’s categories are primarily focussed on student’s coding behaviours, they can only be partially compared to the Programming Learner Profiles. Perkins’ Stopers category can be roughly mapped to the Reluctant Beginners, Willing Beginners and Budding Managers profiles in this study. Budding Managers are different in that they behave as Stopers for the more complex problems. Perkins’ Movers category roughly maps to Keen Beginners, Budding Practitioners, Budding Developers and Advanced Developers. Perkin’s Extreme Movers category roughly maps to the Willing Beginners. Willing Beginners display an ad-hoc and unsystematic approach to coding, similar to Tinkerers.

**Rocket Scientists and Strugglers (Jenkins and Davy)**

Jenkins and Davy (2000) divided their student cohorts into four distinct groups based on their level of programming knowledge: Serious Strugglers, Strugglers, Averages and Rocket Scientists. Their classification was primarily based on their own teaching experiences, while this study derived the classifications from the student interview data.
Jenkins and Davy’s Serious Strugglers roughly map to Reluctant Beginners. Strugglers can be mapped to Reluctant Beginners, Willing Beginners and Budding Managers. Averagers roughly map to Keen Beginners, Budding Practitioners and Budding Managers and Rocket Scientists can be mapped to Budding Developers and Advanced Developers.

Effective and Ineffective Novices (Robins, Rountree and Rountree)

Robins, A, Rountree and Rountree (2003) identified two types of novices: Effective and Ineffective based on the strategies these use to comprehend and generate programs. Their classification is based on a literature review relating to the psychological/educational study of programming while this study derived the classifications from the student interview data.

Robins, Rountree and Rountree’s Ineffective Novices can be mapped to Reluctant Beginners, Willing Beginners and Budding Managers. Robins, Rountree and Rountree’s Effective Novices can be mapped to the Keen Beginners. Budding Practitioners, Budding Developers and Advanced Developers can not be mapped to Robins, Rountree and Rountree’s types because these are not novices but are students in later years with a competent or advanced skill level.

Narratives on Learning to Program (Cardell-Oliver)

Cardell-Oliver (2014) developed narratives to describe student experiences of a first programming course by analysing various artefacts produced by students during teaching. Cardell-Oliver describes four student experiences: Thriving, Surviving, Drowning and Lost.

Cardell-Oliver’s Lost category can be mapped to the Reluctant Beginners in that they are capable but they are unable to adjust their learning strategies to learn programming as a result of their emotional responses to difficulties. Reluctant Beginners are reluctant, however, as a result of not perceiving programming relevant to them, despite emotional responses.
The Drowning category can be mapped to the Willing Beginners and Budding Managers (in some instances) based on the shallow learning strategies they use, avoidance strategies and gaps in their understanding. The Surviving category can be mapped to the Budding Managers because they focus on optimising their performance in assessments. Budding Managers however, leverage their personal networks to compensate for their lack of skill or gaps in their understanding. Cardell-Oliver’s Thriving category map to Keen Beginners, but also describes what Budding Practitioners, Budding Developers and Advanced Developers where like in their first programming course.

7.12 Chapter Summary

This chapter compared and contrasted the theory developed by this study with theories and findings from the literature. This enfolding was done as part of the research plan outlined in chapter 3 and as an important part of the Grounded Theory methodology. This enfolding grounds the theory within the literature by showing where the findings are similar to previous studies and where they are different or new.

The contributions of this study to the body of knowledge have been highlighted in this chapter and are discussed further in chapter 8, namely:

- Theory of Influences on the Student Learning Experience of Programming
- Programming Learner Profiles
- Perceived Personal Relevance
- Patterns of Collaboration
- Patterns of Information Use
- Multiple Mindsets
Chapter 8  Conclusions and Future Research

This study has developed a new Theory of Influences on the Student Learning Experience of Programming that provides a more holistic, deeper and richer understanding of the student programming experience. It encompasses the influences on the student experience both inside and outside the classroom and includes how students interact with their personal networks and how they use information sources, both written and online. The theory developed from a qualitative study that interviewed thirty-one students about their experiences learning programming and analysed the transcripts using a Grounded Theory methodology (Strauss & Corbin 1998).

Learning to program is difficult for many students all over the world and teaching programming is still considered a major challenge (Caspersen & Bennedsen 2007). More recently, researchers have recognised the need for a greater understanding of how students experience learning to program, from the student’s perspective (Bruce et al. 2004). The Theory of Influences on the Student Learning Experience of Programming, developed by this research, contributes to this growing body of knowledge. This theory gives educators a greater insight into what students are thinking and doing when learning to program, both inside and beyond the classroom.

This chapter summarises the research in this study. Firstly, the chapter presents the essence of the Theory of Influences on the Student Learning Experience of Programming. Then, how this theory answers the research questions is discussed. Next, the contributions of this study to the programming education body of knowledge are discussed. Some limitations of the study are then presented. Finally, implications of this research for stakeholders are discussed and suggestions for future research are presented.

8.1  Essence of the Research

The essence of this research is encapsulated in the Theory of Influences on the Student Learning Experience of Programming, which is described in chapters 4, 5 and 6. The theory includes the Model of Influences of Learning Programming (Figure 4-1) and the Model of Dependency (Figure 5-1). These two models are reproduced in Figure 8-1 and a brief description of each component follows.
Figure 8-1 - Essence of the Theory of Influences on the Student Learning Experience of Programming - Model of Influences on Learning Programming and Model of Dependency
The characteristics of the individual students have an influence on their learning. These include general characteristics (age, gender, stage (year) of degree, degree enrolled and working commitments outside of study) and programming context characteristics (preconceived expectations, previous programming studies, prior knowledge and skills in IT, mindset and self-efficacy for programming).

Personal networks are the people students consult or seek information from. These include peers within the same class or year level, friends in the same degree, friends in other degrees, friends in industry and family members. Personal networks also include teaching staff (lecturers, tutors and help lab tutors).

Information Sources students use include written materials (lecture and tutorial notes) as well as online sources such as the Google, online manuals and tutorials, YouTube, professional programming forums and internal course forums.

Core Learning Perspectives describes the key attitudes, intentions and behaviours of students as they proceed with their learning and includes three key concepts: Ownership of Learning, Learning Task Intent and Problem Solving Behaviours. Ownership of Learning describes whether students take responsibility for their learning or assume others are responsible for teaching them. Learning Task Intent describes whether students put priority on understanding or just getting the work done to achieve the marks. Problem solving behaviours describe the various behaviours students will carry out when solving programming problems. These range from being ad-hoc and poorly planned to thorough and thoughtful coding, error detecting and debugging.

Patterns of Collaboration describe the ways students interact and seek assistance. In One Way Dependency, students are highly reliant on their personal networks. In a Two Way Co-dependency, both parties are co-dependent on each other for different skills or division of labour. In the Collaborative Independent interaction, both parties proactively attempt their tasks and then come together to compare approaches. Solitary Independent students help their peers but don’t seek help from their personal networks. Students may interact with multiple peers or personal networks concurrently.

Patterns of Information Use describes the different ways students use written and online sources. At a basic level, students look up their information sources for syntax verification and checking error messages. Looking for Sample Answers involves looking for a section of code that matches the students’ problem. Code Adaptation involves identifying subsections of code written by others that could be adapted. Comparative Analysis involves students comparing their code to others to improve their own coding style. Beyond the Scope involves using online sources to learn new languages or features not covered in the curriculum.
Learner Nature includes three distinct dimensions that influence the way students approach their learning of programming and their learning behaviours: Perceived Personal Relevance, Learning Trait and Skill Level in Programming. Perceived Personal Relevance is the level of relevance the students see in learning programming in relation to their future career goals. Learning Trait is how much students rely on others for solving programming problems and their learning (dependent or independent). Skill Level is their programming skill level, ranging from novice, to competent and advanced.

Programming Learner Profiles describe the influences on the way students go about learning programming and the way they use their information sources and personal networks. The Programming Learner Profiles encapsulate the relationships and influences between the three Learner Nature dimensions (Perceived Personal Relevance, Learning Trait and Skill Level) and the patterns of learning behaviours (Core Learning Perspective, Patterns of Collaboration and Patterns of Information Use).

8.2 Research Questions

This section presents the research questions outlined in chapter 1 and answers each question in light of the theory generated by this study. The research questions of this study were:

What are the key influences on the learning experience of programming students?

The learning experience of programming involves a complex interaction of a wide range of influences. The students themselves bring a range of characteristics that can influence their learning: preconceived expectations, previous programming studies, previous IT knowledge and a mindset and self-efficacy towards programming.

A major influence is the student’s Perceived Personal Relevance towards programming. Perceived Personal Relevance, together with Learning Trait and Skill Level describe the Learner Nature of the student. There is a bi-directional influential relationship between Learner Nature and Learning Behaviours. Within Learning Behaviours, the influences that define Core Learning Perspectives (Ownership of Learning, Learning Task Intent and Problem Solving Behaviours) are closely related and intertwined. Core Learning Perspectives, in turn, influence Patterns of Collaboration and Patterns of Information Use. As Learning Behaviours evolve, these can also have a feedback influence on Learner Nature.

Patterns of Collaboration describe four levels of interaction: One Way Dependent, Two Way Co-Dependent, Collaborative Independent and Solitary Independent. The varying Patterns of Collaboration influence how students interact with and use their Personal Networks and, in turn, students can be influenced by their Personal Networks. Patterns of Information Use describe the different ways students use information sources, ranging from basic syntax verification to more advanced forms of solution discovery. These varying patterns influence how students use and interact with their information sources.

The Programming Learner Profiles encapsulate the relationships and influences between the three Learner Nature dimensions, which define the profiles, and their influences on the Learning Behaviours, and, in turn, the influences of the Learner Behaviours on Personal Networks and Information Sources. Each profile describes, in essence, the nature and behaviour of different types of students. The seven distinct Programming Learner Profiles are: Reluctant Beginner, Willing Beginner, Keen Beginner, Budding Manager, Budding Practitioner, Budding Developer and Advanced Developer.
**What influences motivate students to learn how to program?**

The most influential motivation that arose from this study was that of Perceived Personal Relevance. Students who perceive that programming is relevant to their future career goals are far more motivated to learn it. They have a deep desire to thoroughly understand their learning tasks and willingly do their own exploration to seek understanding. They work independently but collaborate with their trusted personal networks.

On the other hand, the students who do not perceive programming as relevant to them are inclined towards completing the course deliverables rather than learning the concepts. They tend to interact with their personal networks in a dependent fashion. They skip learning tasks and have difficulty understanding the information they find.

**What types of learners do we find when teaching programming?**

Students have a quite diverse range of learning styles when learning programming. Reluctant Beginners do not see programming as relevant to them, at all, and heavily depend on others for their learning. Willing Beginners may not see programming as currently relevant to them but are still willing to learn. Keen Beginners are the students who can see themselves as programmers. They work independently and collaborate with their personal network.

Budding Managers see themselves working in IT in a non-technical role. They have evolved their skills somewhat from Beginners but have also learnt to leverage their social networks to improve their learning outcomes. Budding Practitioners are interested in a technical career that they see programming relevant to, but they don’t want to be programmers. Budding Developers are more evolved Keen Beginners in that they see themselves as programmers and have evolved their skills to a much more competent level.

Advanced developers are those students that have developed their programming skills to that of a professional industry level. Budding and Advanced Developers not only learn very independently but also help other students with their learning.
How do students use their personal networks in learning programming?

Students have wide personal networks they may seek help from: peers in the same class, friends in the same degree, friends studying in different degrees or universities, friends in industry, family members, teaching staff and help lab tutors. The majority of interactions with their personal networks happen outside the classroom and they may consult multiple personal networks for different types of help. When seeking help from staff, students seek the most convenient source at the time, depending on the issue, urgency and time available.

Students vary in how dependent they are on their personal networks for their learning. In a One Way Dependency, the student seeking help is totally dependent on the peer, who acts as a crutch or as delegation service. In a Two Way Co-Dependency, students rely on each other through skill or task based co-dependency. Collaborative Independent students practice a stronger two way interaction or exchange of information with both parties truly complementing each other. Solitary Independent students help and guide other students, but they themselves tend to not seek help from their peers. Students can interact with multiple peers or personal networks concurrently, at different levels.

How do students use information sources in learning programming?

Students use a wide variety of information sources: course notes, books, Google, online manuals, online tutorials, YouTube, external programming forums and internal course forums. At a basic level, all students look up information sources for reference checking (verifying the correct syntax to use) and for finding the meaning of error messages. Novices, however, have difficulty interpreting and using what they find online.

Information sources are also used for various forms of Solution Discovery: Looking for Sample Answers, Code Adaptation, Comparative Analysis and Beyond the Scope. All students look for sample code, but novices tend to look for complete answers online, and have difficulty understanding and using what they find. Code Adaptation involves identifying and understanding parts of subsections of code written by others that can be adapted.

Comparative Analysis involves students comparing their code to others (personal networks, online forums and written materials) with the intention of improving their own coding style. Beyond the Scope involves using online sources to learn new languages or advanced features. These last two are mainly practiced by the more advanced students.
8.3 Contributions of this Research

This section briefly highlights the various contributions of this study to the body of knowledge.

Theory of Influences on the Student Learning Experience of Programming

This study has developed the Theory of Influences on the Student Learning Experience of Programming. While there are previous studies into learning programming, this theory is more comprehensive than previously developed theories. It explores a wide range of influences on the learning experience, from the student’s perspective. While many of these influences have been previously researched, this study brings these influences together and presents them in a cohesive model.

This theory is more holistic than previous theories and models. It encompasses the learning experiences both inside and beyond the classroom. It introduces seven new Programming Learner Profiles, which encompass student’s experiences in all stages of their studies. It includes patterns of how students interact with their personal networks and also how they seek and use information.

Programming Learner Profiles

The Programming Learner Profiles developed in this research are a form of learner categorisation of programming students. Each profile describes, in essence, the nature and behaviour of a different type of student. The seven profiles developed are: Reluctant Beginner, Willing Beginner, Keen Beginner, Budding Manager, Budding Practitioner, Budding Developer and Advanced Developer.

This categorisation is more holistic and comprehensive than previous programming learner categorisations. Whereas most of the previous categorisations were defined along a single concept (predominately skill level), the Programming Learner Profiles were defined by combining Perceived Personal Relevance, Learning Trait and Skill Level. Whereas most of the previous categorisations were limited to first year and novice programming students, the profiles in this study encompass students from all stages in their study.

Programming Learner Profiles also describe the behaviour of students more deeply than previous categorisations, encompassing not only their coding behaviours but also their interactions with their personal networks and their information behaviour.
Perceived Personal Relevance

The most influential motivation that arose from this study was that of Perceived Personal Relevance. This motivation was also one of the key defining concepts of the Programming Learner Profiles. How students perceive programming as being relevant to their future career goals arose as a key influence over many of the behaviours students exhibit in their learning. This influence has not previously been highlighted in programming education studies.

Patterns of Collaboration

The Dependency Model developed as part of the Patterns of Collaboration theme describes four levels of dependency students have with their Personal Networks: One Way Dependent, Two Way Co-Dependent, Collaborative Independent and Solitary Independent. While previous studies have made reference to some aspects of these interactions, this model provides a new cohesive perspective on how students may interact with their peers that has not previously been reported in the literature.

Patterns of Information Use

Patterns of Information Use describe various ways students use their information sources: Verifying Syntax, Finding the Meaning of Errors, Looking for Sample Answers, Code Adaptation, Comparative Analysis and Beyond the Scope. While many previous studies have detailed the problems students encounter and their problem solving behaviours in class, this categorisation provides a new perspective on the different ways students use their information sources to solve programming problems in a wider context, both inside and beyond the classroom.

Mixed or Multiple Mindsets

While there is considerable research into the concepts of fixed and growth mindsets, the concept of mixed or multiple mindsets that arose in this study - that students can have a fixed mindset towards one topic, and at the same time, have a growth mindset towards another topic - has not been given much attention in the mindset literature. This study can therefore potentially make a contribution to further the understanding of student mindsets.
8.4 Implications of this Research

This section considers the implications that the findings of this study may have to programming educators, the Information Systems community and researchers.

The comprehensive and holistic theory developed in this thesis provides teachers a deeper understanding of the student programming learning experience. This can provide teachers greater insight into how their students are actually learning and encourage them to consider the student perspective as an important part of the learning process. As well as the theory as a whole giving teachers a better understanding of their students, individual aspects of the theory can also give teachers ideas as to how they might design their curriculum, delivery and assessment.

Perceived Personal Relevance arose during the analysis as a major motivation for programming students. Understanding the importance of this motivation can have implications to teaching practice in a number of ways. Firstly, teachers should make it clear to students how the programming they are learning will be relevant to them in the future, even if their career goals are not be a programmer. This could involve highlighting the various potential careers students may take up, and how knowledge of programming would be relevant to these careers. Teachers can also ensure that the programming curriculum is relevant to a number of career contexts. Tutorial examples and assignments, for example, could be based around scenarios that are relevant to multiple career goals, instead of abstract activities. Finally, teachers could also make efforts to better understand the career goals of each cohort of students. This might involve asking students to fill out a short survey at the beginning of semester, asking them what their future career goals are and their pre-conceived expectations of learning programming.

Another key aspect of the theory was the development of Programming Learner Profiles. Teachers could benefit greatly from a better understanding of the different types of student they might be dealing with and how each type of student has different goals and learning styles. Rather than the ultimate goal being to develop every student into a professional programmer, teachers should consider the various pathways students might take in their learning profiles and that there are many possible successful outcomes for students, depending on their goals. While a high level of programming competence would be a successful outcome for a Budding Developer, this may not necessarily be the case for a Budding Manager.
If teachers design curriculum, tasks and assessment with multiple Programming Learner Profiles in mind, the curriculum will be more likely to be relevant and achievable for a larger proportion of the students. If teachers are aware of the variety of learner they may be dealing with, they are more able to tailor assistance to each student in the classroom, instead of using the same approach for all students.

The Patterns of Collaboration theme within the theory highlights the various levels of dependence that students may have on others in their learning. It also highlights the wide range of personal networks that students make use of when seeking help with programming problems. Having a greater understanding of these student behaviours would make teachers more aware that student collaboration extends beyond the classroom. While there have been initiatives to encourage student collaboration within the classroom, such as group work and pair programming, further initiatives are needed to encourage true collaboration outside of the classroom. Students should be encouraged towards being collaborative independents, using their personal networks to compare and contrast their work, rather than seeking solutions in a dependent fashion. The issue of one way dependence should also be taken into consideration by those implementing student mentoring type of interventions, as the students seeking help risk becoming too dependent on their helpers.

Patterns of Information Use also provide deeper insights into how students use information sources and online forums. The fact that a large amount of the student learning happens outside the classroom should be embraced by teachers. Whilst all students use Google, this study highlights that novices have difficulty finding and using information online. Rather than leaving students to discover these methods by trial and error, programming educators should provide novice students with classroom activities that explicitly demonstrate efficient methods of searching for sample code using Google and how to use the code that is found. Students could also be shown how to use online resources to verify code syntax and for searching for the meaning of errors, including how to interpret the results of the search. Similarly, students could be shown how to search online forums for help with coding problems and how to interpret the posts they find. The various forms of solution discovery outlined in Patterns of Information can also form the basis of student instruction. Students could be given classroom tasks that show them how to search for sample code online and incorporate found code into their own programs. They could also be given tasks to find code online and compare and contrast it to their own code.
The findings of this study also have implications to other programming education researchers. This study highlights that students do a considerable amount of their learning outside of the classroom. Researchers need to consider this broader learning environment when designing research projects. Research based solely on classroom activities may provide a narrow picture of the learning process. In a similar vein, this study also highlights that students use many online resources in their learning. Research that focuses solely on materials provided to the students may also provide a limited view of the complete learning process. Future researchers need to consider all of the information sources that student may make use of. The Programming Learner Profiles can also provide researchers with an understanding that students can have very different goals and learning styles, and these need to be considered in research design. Future researchers should further explore the different types of programming learners and not assume all students learn in the same way.

Lastly, this study has implications to the field of Information Systems. This study highlights that Information Systems students have a wide range of career goals in mind during their learning, which tends to mirror the field itself, which embraces a wide range of industry roles. Programming should be seen as important and relevant to all potential Information Systems industry roles, not just programmers. Other disciplines are now embracing programming as relevant to them, and the discipline of Information Systems can play a part in enabling computing skills to be relevant to a wider population of people.
8.5 Limitations of this Research

This study had some limitations which could potentially have affected the outcomes. In order to gain deeper insights by analysing rich data, it was necessary to interview a relatively small number of students. Also, due to practical and ethical constraints, it was not possible to recruit and interview students who had dropped out from their degree, as there was no way of communicating with them once they ceased being students of the university.

The theory generated is based on the experiences of the 31 students that participated in this study. The theory can be analytically generalised (Yin 2003), because it is grounded in both the data and the literature. It therefore has the potential to explain the influences on and behaviours of other programming students and could also be applicable in other education contexts. Statistical generalisation to the general population of all programming students, however, is not possible, due to the limited sample size.

Due to the qualitative nature of this study, there was the potential of researcher bias that could influence the analysis process and findings. The researcher was always aware of this potential and sought to minimize researcher bias by following a highly iterative research process that developed theory that was grounded in the data. This is further discussed in section 3.6.4.

This study was conducted in Australia and all of the students interviewed were either current or past students of RMIT University. The findings may have been different if the study had been conducted in another university, country or culture. Having said that, many of the themes that arose in this Grounded Theory study were consistent with the themes from other studies conducted around the world.
8.6 Suggestions for Future Research

There are a number of possible directions for future research based on this study.

Firstly, the theory developed by this study could be tested with a larger population of students using a survey. While the theory developed in this study can be analytically generalised to other groups of students, it cannot be statistically generalised to represent the behaviour of all programming students. A survey would allow this question to be explored. This could initially involve surveying student cohorts from within the same Information Systems degree at RMIT University where this study was conducted, but could also extend to surveying students in other degrees and/or other universities. A survey could explore whether key aspects of the theory are applicable to all programming students. Are the seven Programming Learner Profiles developed in this study applicable to all programming students? Is Perceived Personal Relevance an important influence on the motivation of all students as to how much effort and focus they devote to learning programming? Are the different levels of dependence within personal networks found to exist in other populations of programming students? Do all students exhibit the same patterns of information use as the students in this study?

Another direction would be to investigate, in depth, the experiences of another group of programming students. A group of students in a different degree or from a different university could be interviewed. Alternatively, a group of high school students learning programming could be interviewed. The analysis results could be compared with those from this study. This would help to further develop and refine the theory analytically and make it more relevant to a wider population of students.

A third direction would be to perform a similar, in depth, study but focussing on a different topic within IT or in a different discipline. The goal would be to see how much of the theory is transferrable to other educational contexts. For example, the study might involve looking at the influences and behaviours of students learning databases/SQL, systems analysis or networking. Alternatively, the study could involve interviewing a group of accounting students about how they go about learning computerised accounting systems.
Chapter 9  Appendix

9.1 Interview Questions

1. Personal Background - Can I ask your age group and occupation?
   i. Age Group (18-29, 30-39, 40+), Sex (M/F), Occupation, Degree
2. Previous Experiences
   a. What studies did you do before this degree?
   b. Tell me about your previous experiences with programming.
3. Experiences (Stages: Problem, Planning, Coding, Debugging, Testing)
   a. Can you describe your experiences of programming in your course(s)?
   b. What aspects did you enjoy?
   c. Can you tell me what you found easy and what you found hard & why?
      - What aspect was hard about it?
      - How did you try to work out the problem?
   d. What did you find frustrating?
      At What Point? / What stage? What did you do?
   e. What aspects did you find stressful? - What made you feel stressed?
      What did you do?
   f. Did you have enough time to learn?
      i. How much time did you spend on it?
      ii. What other demands do you have on your time?
   g. How does time you spend in programming compare with other subjects?
   h. How do you feel about programming now? Why?
   i. How does programming compare to your other courses?
4. Seeking Help
   a. When you have a problem, where do you seek help?
   b. Why do you seek help from <X> first?
   c. How long do you try to figure it out yourself before asking for help?
   d. Do you work with others in a group?
   e. Tell me how you use Google.
5. Expectations
   a. What did you expect to learn in your degree?
   b. Was the programming you learnt what you expected?
   c. Do you see programming being relevant in your future?
   d. What have others told you about programming?
6. Mindset
   a. Can you give me an example of a difficulty you faced and how you dealt with it?
   b. How did you deal with bugs?
   c. Do you think you were successful in learning programming?
      i. How do you gauge that success?
   d. Do you think your mark reflected your efforts?
7. What is your approach to learning programming?
8. What advice would you give first year students who are learning programming?
9.2 Ethics Plain Language Statement

INVITATION TO PARTICIPATE IN A RESEARCH PROJECT
PROJECT INFORMATION STATEMENT

Project Title:
- Factors influencing the attitudes and perceptions of information systems students towards programming.

Investigators:
- Fran Adamopoulos (Phd Student, Lecturer, School of BIT & L, RMIT University, fran.adamopoulos@rmit.edu.au, 9925-1375)
- Dr Martin Dick (Supervisor, Lecturer, School of BIT & L, RMIT University, martin.dick@rmit.edu.au, 9925-6075)

Introduction
You are invited to participate in a research project being conducted by RMIT University in the School of Business Information Technology. This information sheet describes the project in straightforward language, or ‘plain English’. Please read this sheet carefully and be confident that you understand its contents before deciding whether to participate. If you have any questions about the project, please ask one of the investigators.

Who is involved in this research project?
This research is being conducted as part of a PhD program which is being undertaken by Fran Adamopoulos. This research project has been approved by the Portfolio Human Research Ethics Sub Committee.

Why is it being conducted?
This project is attempting to explore what factors influence the attitudes and perceptions of information systems students towards programming. The data elicited will then be used, together with current information systems research obtained from the literature, to develop a model that will attempt to explain which factors, how and why they impact and influence student attitudes and perceptions.

Why have you been approached?
Participants in this research have been invited to volunteer some time to discuss their experiences in learning programming. You have responded to a general request for participants that was made to students currently enrolled in the Bachelor of Business in Business Information Systems degree or that have completed the degree.

What is the project about? What are the questions being addressed?
The primary research question is: “What are the factors that influence the attitudes and perceptions of information systems students towards programming?”

The research is looking to explore which factors motivate or deter students from engaging in the study of programming. Which phase of programming is impacted most by the factors and why? It is also hoped that some guidelines can be drawn as to what can be done to improve the study/teaching of programming.

It is intended that approximately 20 to 30 students will be interviewed.
if I agree to participate, what will I be required to do?

You will be asked to participate in a semi-structured interview, for approximately 1 hour. You will be asked to provide some background details about yourself and your attitudes, perceptions and experiences with programming in your studies.

You may choose not to answer any particular question. This interview will be recorded (audio only) and you (the participant) have the right to request that recording cease at any stage during the interview.

By agreeing to participate in this research, you acknowledge that your transcript of results may be viewed as part of this research process.

What are the risks or disadvantages associated with participation?

There are no apparent risks in participating in this research as it only involves a discussion of your perceptions, experiences and study activities towards programming. If any questions may cause you concern, you are free not to answer them. If you (the participant) are unduly concerned about your responses to any of the interview questions or discussions or if you find participation in the interview distressing, you should advise the interviewer, that you either want to strike that discussion from the record or discontinue the interview. The researchers will discuss your concerns with you confidentially and suggest appropriate follow-up, if necessary.

What are the benefits associated with participation?

This research hopes to benefit the field of psychological/educational study of programming and also provide some guidelines to improve the teaching of programming as well as student’s successful completion of their programming courses. The researchers are happy to make available to you, the participant, any results, papers, and other outcomes from this research.

What will happen to the information I provide?

All recorded data will be transcribed, encrypted and archived. The transcribed data will be kept during the analysis phase of the research on the primary researcher’s laptop computer and will be stored offsite at the primary researcher’s residence. The encrypted data will be backed up onto a USB storage device and CD media storage. One backup copy will be stored securely at RMIT in the School of Business Information Technology and the other backup copy will be kept offsite at primary researcher’s residence.

All the data will be kept for 5 years upon completion of the project, after which it will be destroyed.

The interview data will be treated in a strictly confidential way and will only be viewed by the researchers involved in this project. Any outcomes from this research will be of a general nature without any details of specific participants disclosed. Where a participant’s words are directly quoted in a publication, it will be with absolute anonymity.

Any information that you provide can be disclosed only if (1) it is to protect you or others from harm, (2) a court order is produced, or (3) you provide the researchers with written permission. If the data is required for some other purpose (other than use in this project), then permission will be obtained from the participants before use.

What are my rights as a participant?

You have the right to withdraw your participation at any time, without prejudice. You have the right to have any unprocessed data withdrawn and destroyed, provided it can be reliably identified and it does not increase the risk for the participant. Participants also have the right to have any questions, in relation to the project and their participation, answered at any time.
Whom should I contact if I have any questions?
The primary investigator (Fran Adamopoulos – fran.adamopoulos@rmit.edu.au or 03 99251375)
or her supervisor (Dr. Martin Dick – martin.dick@rmit.edu.au or 03 99255976) should be
contacted, contact details given previously.

Yours Sincerely

Fran Adamopoulos  
Master of Computing

Dr. Martin Dick  
Doctor of Philosophy in Computing

Any complaints about your participation in this project may be directed to the Secretary, Portfolio Human Research Ethics Sub Committee, 
Business Portfolio, RMIT, GPO Box 2476V, Melbourne, 3001. The telephone number is (03) 9925 5894 or email address rcu@rmit.edu.au. 
Details of the complaints procedure are available from the above address or http://www.rmit.edu.au/council/hec.
9.3 Participant Consent Form

RMIT HUMAN RESEARCH ETHICS COMMITTEE
Prescribed Consent Form for Persons Participating in Research Projects Involving Interviews, Questionnaires, Focus Groups or Disclosure of Personal Information

PORTFOLIO OF
SCHOOL/CENTRE OF

Name of Participant:

Project Title:
Factors influencing the attitudes and perceptions of information systems students towards programming

Name(s) of Investigators: (1) Fran Adamopoulos Phone: (03) 9925 1375
(2) Dr Martin Dick Phone: (03) 9925 5976

1. I have received a statement explaining the interview involved in this project.
2. I consent to participate in the above project, the particulars of which - including details of the interview - have been explained to me.
3. I authorise the investigator or his or her assistant to interview me.
4. I give my permission to be audio taped: ☐ Yes ☐ No
5. I acknowledge that:

(a) Having read the Plain Language Statement, I agree to the general purpose, methods and demands of the study.
(b) I have been informed that my identity will not be revealed by this research.
(c) I have been informed that I am free to withdraw from the project at any time and to withdraw any unprocessed data previously supplied.
(d) The project is for the purpose of research and/or teaching. It may not be of direct benefit to me.
(e) My transcript of results may be viewed as part of this research project.
(f) The privacy of the information I provide will be safeguarded. However should information of a private nature need to be disclosed for moral, clinical or legal reasons, I will be given an opportunity to negotiate the terms of this disclosure.
(g) The security of the research data is assured during and after completion of the study. The data collected during the study may be published, and a report of the project outcome will be provided to participants on request. Any information which may be used to identify me will not be used unless I have given my permission (see point 5).

Participant’s Consent

Name: ____________________________ Date: ____________________________

(Participant)

Name: ____________________________ Date: ____________________________

(Witness to signature)

Participants should be given a photocopy of this consent form after it has been signed.

Any complaints about your participation in this project may be directed to the Chair, Human Research Ethics Sub-Committee, Business Portfolio, GPO Box 2476V, Melbourne, 3001. The telephone number is (03) 9925 5594 or email address rlu@rmit.edu.au. Details of the complaints procedure are available from www.rmit.edu.au/council/hrec.
9.4 Ethics Approval

Ref: Ethics Appl. 1000107

Tuesday, March 30 2010

Francisca Adamopoulos
School of Business Information Technology and Logistics
College of Business
RMIT University

Dear Francisca,

I am pleased to advise that your application for ethics approval for a Research Project has been approved by the Chair of the Business College Human Ethics Advisory Network. Approval has been granted for the period from 24 March 2010 to 16 July 2015.

The RMIT Human Research Ethics Committee (HREC) requires the submission of Annual and Final reports. These reports should be forwarded to the Business College Human Ethics Advisory Network Secretary. Annual Reports are due in December for applications submitted prior to September the year concerned. I have enclosed a copy of the Annual/ Final report form for your convenience. Please note that this form also incorporates a request for extension of approval, if required.

Best wishes for your research.

Yours sincerely,

Kristina Tsculis-Reay
Secretary
Business College Human Ethics Advisory Network

Encl.
RMIT BUSINESS
COLLEGE HUMAN ETHICS ADVISORY NETWORK (BCHEAN)

Application for Approval of Research Project

SUMMARY & APPROVAL

Project Title: Factors Influencing the Attitudes and Perceptions of Information Systems Students towards Programming

Principal Investigator: Francisca Adamopoulos

Supervisor: Martin Dick

Project Category: 2

Degree for which research is undertaken (if applicable): PhD

School Name: Business Information Technology and Logistics

Contact Telephone Number: 9925 1375

Email Address: fran.adamopoulos@rmit.edu.au

BUSINESS COLLEGE HUMAN ETHICS ADVISORY NETWORK USE ONLY:

Date Application Received: 30 November 2009

Business College Human Ethics Advisory Network Register No: 1000107

Period of Approval: 24 March 2010 to 16 July 2015

Comments / Provisos: N/A

The Business College Human Ethics Advisory Network assessed the Project as Category 2

Signature: [signature]

Date: 24 March 2010

Associate Professor Adela McMurray BCHEAN Chair
9.5 Coding in NVivo Examples

Figure 9-1 shows the list of sources (interviews) coded with total number of nodes and references for each interview.

![Figure 9-1 - List of sources (interviews) coded]

Open Coding Examples

Figure 9-2 below shows a list of all the high level nodes (concepts) generated through the open coding analysis phase.

![Open Coding Examples](image-url)
The images below show the top level node (concept) for Learning with sub concepts at two different points in time during the analysis. The Fundamental Learning Perspective sub-concept, in Figure 9-4, was later further evolved and renamed to Core Learning Perspective, which is one of the concepts within Learning Behaviours in the final model.
Earlier top level (node) concept Learning with its sub concepts

Evolved top level concept Learning with its sub concepts

Figure 9-3 Earlier List of the Learning Concept

Figure 9-4 Evolved List of the Learning Concept
In Figure 9-6 below, the concept “Sources of Help” later evolved into “Information Sources”. The subconcept “Patterns of Information Use” was later promoted and became part of the final model.

![Figure 9-5 Earlier list of Sources of Help](image)

![Figure 9-6 Evolved list of Sources of Help](image)

In Figure 9-8 below, the concept “Social Networks” later evolved into “Personal Networks”. The subconcept “How they collaborate” was later promoted and renamed to “Patterns of Collaboration”.

![Figure 9-7 Earlier list of Social Networks](image)

![Figure 9-8 Evolved list of Social Networks](image)
Axial Coding Examples

The following Axial coding examples (Table 9-1 and Table 9-2) display the relationships between the main concepts generated from the open coding analysis done in NVivo. These Axial coding examples display a preliminary description of the Willing Beginner and Budding Manager profiles and were done in Excel.

Table 9-1 Preliminary description of the Willing Beginner Profile

Table 9-2 Preliminary description of the Budding Manager Profile
Axial Coding – Early Summary of the Programming Learner Profiles

The summary table below, Table 9-3, displays an evolved summary of the Axial Coding preliminary profile descriptions shown on the previous page. At this point the key concepts were selected but later these were re-grouped and abstracted into higher concepts with different names.

<table>
<thead>
<tr>
<th>Programming Learner Profile</th>
<th>Reluctant Beginner</th>
<th>Willing Beginner</th>
<th>Keen Beginner</th>
<th>Budding Manager</th>
<th>Budding Practitioner</th>
<th>Budding Developer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value in Learning Progress</td>
<td>Low</td>
<td>Low to Intermediate</td>
<td>High</td>
<td>Low</td>
<td>Intermediate</td>
<td>High</td>
</tr>
<tr>
<td>Motivations</td>
<td>Extrinsic</td>
<td>Extrinsic</td>
<td>Intrinsic</td>
<td>Extrinsic</td>
<td>Extrinsic</td>
<td>Extrinsic</td>
</tr>
<tr>
<td>Skill Level</td>
<td>Novice</td>
<td>Novice Intermediate</td>
<td>Novice Top End</td>
<td>Competent enough</td>
<td>Competent</td>
<td>Very Competent</td>
</tr>
<tr>
<td>Learning Trait</td>
<td>Dependent</td>
<td>Independent</td>
<td>Independent</td>
<td>Dependent</td>
<td>Independent</td>
<td>Independent</td>
</tr>
<tr>
<td>Affective Feelings</td>
<td>Dislike To Tolerate</td>
<td>No Interest to Tolerate</td>
<td>Interest</td>
<td>Tolerate</td>
<td>Interest</td>
<td>Very Interested</td>
</tr>
</tbody>
</table>

**Table 9-3 Early summary of the Programming Learner Profiles**
9.6 Matrix Query Examples

The following examples of matrix queries illustrate a different way that the coded data was looked at during the analysis. In the matrix queries below, intermediate concepts (sub concepts) from the top level concepts Sources of Help, Learning, Engagement and Perceptions were selected to be compared with other sub concepts. The matrix query output displays the number of times the coding of a concept overlaps the coding of each of the other concepts selected in the query. A stronger shade of blue indicates a potentially stronger relationship between the intersecting concepts and serves as a guide to look at that specific group of isolated quotes together.

Use of Online Information Sources

Analysis of the quotes produced by the matrix query “Use of Online Information Sources”, shown in Table 9-4, led to the identification of the three key defining concepts (motivation, dependency and skill level) which later led to the development of the Programmer Learner Profiles.

Patterns of Information use

Table 9-5 below shows a more detailed comparison of sub concepts within Patterns of Information Use and Dependency.
Table 9-6 shows a more exhaustive intermediate concept comparison between the Sources of Help sub concepts - Patterns of Information use and Motivations.

Table 9-6 Intermediate concept comparison between Patterns of Information Use and Motivations

**Ownership and Engagement**

Table 9-7 displays a comparison of the intermediate concept Strong Ownership from the Learning top level concept with the sub concept Positive Behavioural Engagement from the Engagement top level concept.

Table 9-7 Intermediate concept comparison between Strong Ownership and Positive Behavioural Engagement
Table 9-8 displays a comparison of the intermediate concepts Weak ownership, Task Goal and Problem Solving (from the Learning Top level concept) with Behavioural, Cognitive and Emotional Engagement.

Table 9-8 Intermediate concept comparison of Weak Ownership and all Engagement types

Conceptual Difficulties

Table 9-9 Intermediate concept comparison of Conceptual Difficulties with itself
References


Beise, C, VanBrackle, L, Myers, M & Chevli-Saroq, N 2003, 'An examination of age, race, and sex as predictors of success in the first programming course'.

Bennedsen, J 2003, 'Teaching Java programming to media students with a liberal arts background', paper presented to Proceedings for the 7th Java & the Internet in the Computing Curriculum Conference (JICC 7) London, United Kingdom.


---- 2005b, 'Programming: Factors that Influence Success', paper presented to SIGCSE’05, St. Louis, Missouri, USA, February 23–27.


Blaho, M, Fodrek, P, Farkas, L & Foltin, M 2013, Preferred information sources and the quality of online sources for programming courses.

Boekaerts, M 2016, 'Engagement as an inherent aspect of the learning process', Learning and Instruction, vol. 43, pp. 76-83.


Carter, J, Bouvier, D, Cardell-Oliver, R, Hamilton, M, Kurkovsky, S, Markham, S, McClung, OW, McDermott, R, Riedesel, C, Shi, J & White, S 2011, 'Motivating all our students?', paper presented to Proceedings of the 16th annual conference reports on Innovation and technology in computer science education - working group reports, Darmstadt, Germany.

Caspersen, ME & Bennedsen, J 2007, 'Instructional design of a programming course: a learning theoretic approach', paper presented to Proceedings of the third international workshop on Computing education research, Atlanta, Georgia, USA.


Cope, C 2000, Educationally critical aspects of the experience of learning about the concept of an information system, La Trobe University, Bendigo.


Creswell, JW 1998, 'Qualitative inquiry and research design: choosing among five traditions'.

305


Davis III, CH, Deil-Amen, R, Rios-Aguilar, C & Gonzalez Canche, MS 2012, 'Social media in higher education: A literature review and research directions', Report printed by the University of Arizona and Claremont Graduate University, vol. 8.


Duncan, EN 2006, 'Learning computer programming in Irish third-level institutions: a study of first year students' experiences', Education.


Eccles, JS 2016, 'Engagement: Where to next?', *Learning and Instruction*, vol. 43, pp. 71-5.

Edwards, SL 2005, 'Panning for gold: influencing the experience of web-based information searching'.

Edwards, SL & Bruce, C 2002, 'Needles, haystacks, filters and me: the IT confidence dilemma'.


Facey-Shaw, L, Börner, D, Specht, M & Bartley-Bryan, J 2015, 'A Moodle-based Badge System for Evaluating the Motivational Levels of Introductory Programmers'.


Hagan, D & Markham, S 2000, 'Does it help to have some programming experience before beginning a computing degree program?', paper presented to Proceedings of the 5th annual SIGCSE/SIGCUE ITiCSEconference on Innovation and technology in computer science education, Helsinki, Finland.


Hewner, M & Guzdial, M 2008, 'Attitudes about Computing in Postsecondary Graduates', paper presented to ICER’08, Sydney Australia, September 6-7.


---- 2004, 'The experience factor in early programming education', paper presented to Proceedings of the 5th conference on Information technology education, Salt Lake City, UT, USA.


---- 2015b, 'Modeling programming learning in online discussion forums', paper presented to SciTePress.


Jadud, MC 2006, 'Methods and tools for exploring novice compilation behaviour', paper presented to Proceedings of the second international workshop on Computing education research, Canterbury, United Kingdom.

Jadud, MC & Dorn, B 2015, 'Aggregate Compilation Behavior: Findings and Implications from 27,698 Users', paper presented to Proceedings of the eleventh annual International Conference on International Computing Education Research, Omaha, Nebraska, USA.

Jenkins, T 2001, 'The Motivation of Students of Programming', paper presented to Annual Joint Conference Integrating Technology into Computer Science Education archive Proceedings of the 6th annual conference on Innovation and technology in computer science education Canterbury, United Kingdom


Kanaparan, G 2016, 'Self-efficacy and engagement as predictors of student programming performance: An international perspective'.


Kinnunen, P, McCartney, R, Murphy, L & Thomas, L 2007, 'Through the Eyes of Instructors: A Phenomenographic Investigation of Student Success', paper presented to ICER’07, Atlanta, Georgia, USA, September 15–16.


---- 2011, 'CS majors' self-efficacy perceptions in CS1: results in light of social cognitive theory', paper presented to Proceedings of the seventh international workshop on Computing education research, Providence, Rhode Island, USA.


Klein, HK & Myers, MD 1999, 'A set of principles for conducting and evaluating interpretive field studies in information systems', *MIS Quarterly*, vol. 23, no. 1, pp. 67-93.


Lu, Y & Hsiao, IH 2016, 'Seeking Programming-related Information from Large Scaled Discussion Forums, Help or Harm?', paper presented to The 9th International Conference on Educational Data Mining - EDM 2016, Raleigh, North Carolina, USA.


Lykke, M, Coto, M, Mora, S, Vandel, N & Jantzen, C 2014, 'Motivating programming students by Problem Based Learning and LEGO robots', paper presented to Global Engineering Education Conference (EDUCON), 2014 IEEE.

Maleko, M 2014, 'The mobile social learning environment for novice programmers'.


Perkins, DN, Schwartz, S & Simmons, R 1988, 'Instructional strategies for the problems of novice programmers'.


Postner, L & Stevens, R 2005, 'What resources do CS1 students use and how do they use them?', *Computer Science Education*, vol. 15, no. 3, pp. 165-82.


313


---- 2014, 'Investigating the effects of personality traits on pair programming in a higher education setting through a family of experiments', Empirical Software Engineering, vol. 19, no. 3, pp. 714-52.


Sarkar, A, Lance, M, Lopez, M, Oliver, R & Xu, L 2013, 'Student mistakes in an introductory programming course', paper presented to proceedings of the 2013 conference of Computing and Information technology research and Education in New Zealand incorporating the 26th Annual conference of the National Advisory Committee on Computing Qualifications.


Shaw, R-S 2012, 'A study of the relationships among learning styles, participation types, and performance in programming language learning supported by online forums', *Computers & Education*, vol. 58, no. 1, pp. 111-20.

---- 2013, 'The relationships among group size, participation, and performance of programming language learning supported with online forums', *Computers & Education*, vol. 62, pp. 196-207.


Shi, J 2016, 'Learning to program in a connected world: a study of 1st year undergraduates', University of Southampton.


Swacha, J & Baszuro, P 2013, 'Gamification-based e-learning platform for computer programming education', paper presented to X World Conference on Computers in Education.

Teague, D 2015, 'Neo-Piagetian theory and the novice programmer'.

315


Tew, AE, McCracken, WM & Guzdial, M 2005, 'Impact of alternative introductory courses on programming concept understanding', paper presented to Proceedings of the first international workshop on Computing education research, Seattle, WA, USA.


Tsai, TI 2010a, 'Information horizons of Taiwanese graduate students', paper presented to 2010 iConference Proceedings.


Tsai, TI & Kim, KS 2013, 'Information horizons of college students: Source preferences and source referrals in academic contexts', Proceedings of the Association for Information Science and Technology, vol. 50, no. 1, pp. 1-5.


VCAA 2014, AusVels ICT Curriculum, Victorian Curriculum and Assessment Authority, Victoria, Australia.


Watson, C, Li, FWB & Godwin, JL 2014, 'No tests required: comparing traditional and dynamic predictors of programming success', paper presented to Proceedings of the 45th ACM technical symposium on Computer science education, Atlanta, Georgia, USA.


Wiedenbeck, S 2005, 'Factors affecting the success of non-majors in learning to program', paper presented to Proceedings of the first international workshop on Computing education research, Seattle, WA, USA.


Woszczynski, AB, Haddad, HM & Zgambo, AF 2005a, 'An IS student’s worst nightmare: Programming courses', paper presented to 8th annual Southern Association for Information Systems (SAIS), Savannah, GA.

---- 2005b, 'Towards a model of student success in programming courses', paper presented to Proceedings of the 43rd annual Southeast regional conference - Volume 1, Kennesaw, Georgia.

Yadin, A 2014, 'INDIVIDUALTED ASSIGNMENTS FOR IMPOSING BETTER LEARNING HABITS IN THE” SHARE EVERYTHING” GENERATION', paper presented to The International Scientific Conference eLearning and Software for Education.


