SELF-EFFICACY AND ENGAGEMENT AS PREDICTORS OF STUDENT PROGRAMMING PERFORMANCE: AN INTERNATIONAL PERSPECTIVE

BY

GEETHA KANAPARAN

A thesis

submitted to the Victoria University of Wellington

in fulfilment of the requirements for the degree of

Doctor of Philosophy

Victoria University of Wellington

2016

Abstract

High attrition and failure rates are a common phenomenon in introductory programming courses and are a major concern since course instructors are not able to successfully teach novice programmers the fundamental concepts of computer programming and equip them with skills to code solutions to programming problems. Existing solutions that attempt to minimise the high failure and attrition rates have had little impact on improving the performance of the novice programmers. However, the behaviour of the novice programmer has received little attention from introductory programming course instructors although the literature on learning theory suggests that self-efficacy and engagement are two behavioural factors that affect a student's performance. This study fills the gap in existing research by examining the effect of programming self-efficacy on the engagement of novice programmers, and the effect of their engagement on their programming performance.

A research model that proposes a link between programming self-efficacy and the indicators of engagement that are specific to the context of introductory programming courses, and a link between the indicators of engagement to the programming performance of the novice programmer was developed. A three-phased mixed methods approach which consists of two survey questionnaires and focus groups was used to validate the research model. Data was collected in New Zealand and in Malaysia with 433 novice programmers participating in the survey questionnaires while 4 focus groups were held to refine and validate the indicators of engagement in introductory programming courses. The findings of the focus groups confirmed that participation, help-seeking, persistence, effort, deep learning, surface learning, trial and error, interest, and enjoyment were indicators of engagement while gratification emerged as a new indicator of engagement in introductory programming courses.

The data from the survey questionnaires were analysed using Partial Least Squares Structural Equation Modeling (PLS-SEM). This study found that the programming self-efficacy beliefs of novice programmers had a strong influence on their engagement behaviour with the exception of help-seeking, while effort, enjoyment, deep learning, and surface learning were predictors of programming performance. These findings have implications for introductory programming course instructors and the recommendations emerging from this study include making clear behavioural expectations, designing courses which stimulate and support effective behaviour, and making novice programmers aware of the engagement behaviour that does not lead to better programming performance. This study contributes to the theory of teaching computer programming, and to the practice of designing and delivering introductory programming courses.

Acknowledgements

The journey to obtain my PhD would not have been possible without the love, support, and guidance from my supervisors, family, friends, and acquaintances.

First and foremost, my deepest appreciation goes to my supervisors, Emeritus Professor Rowena Cullen and Dr David Mason. Your responsiveness and professionalism was outstanding. Thank you for your guidance and support.

My Appa and Amma, thank you for your constant encouragement, love and support. My success is as much yours as it is mine.

To my mother-in-law and father-in-law, thank you for your prayers, patience, and warmth.

To my loving husband Gobi, thank you for holding the fort and for believing in me. Thank you too for always showing me the lighter side of life.

To my PhD and non-PhD mates in Wellington: Alok, Balsam, Diane, Donelle, Fahimi, Fiza, Kamlini, Marta, Nicole, Rashidah, Siva, Sivashankar, Syahida and Van, thank you very much for always being there for me, and for making my stay in Wellington a memorable one.

To my friends back home, Malar, Rames, Jaswin and Shamini, thank you for constantly encouraging me.

To my nephew Vishnu, and to my niece Lavenya, thank you for being my sunshine.

Lastly, special thanks to Adrienne McGovern-Wilson, Dr Andrew Seddon, Dr Andrew Luxton-Reilly, Asma'a Mahfoud, Associate Professor Brendan McCane, Corinne Gower, Dr Dalice Sim, Dr David Skelton, Ivy Yeo, Janak Adhikari, Dr Janette Hamilton-Pearce, Julian Wong, Mary Ting, Mick Jays, Murray Wills, Mozhgan, Nathan Humphrey, Dr Paul Dechering, Dr Peter Andreae, Phua YT, Seetha, Seyed Amir, Sharon Lee, Sumaira, and Usha Varatharaju. I am grateful for your generosity and support.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1 Motivation for the research

The demand for software developers has been increasing tremendously in recent years due to the need to develop and maintain software which is the infrastructure behind many business operations today. According to the U.S. Department of Labour, employment for software developers is expected to increase by 22% from 2012 to 2022 (Bureau of Labour Statistics, 2014). This increase is much faster than the average trends reported in other occupations. In another study, Michael Page, a specialist recruitment company revealed that software engineers and software developers are the most in demand professionals in 24 countries including Australia and New Zealand (Michael Page, 2015).

The increasing demand for software developers places pressure on higher educational institutions (HEIs)[1] to produce graduates who are equipped with skills that are relevant to enter the workforce in the software industry. Hence, HEIs are tasked with attracting and retaining students in Computer Science related degrees. However, in the last two decades, researchers and course instructors have been increasingly concerned about the seemingly high failure and attrition rates in the introductory programming courses that are offered at HEIs. This poses a problem to HEIs in meeting the demands of a skilled workforce in the software industry.

Introduction to programming is a core course in the first year of an Undergraduate Computer Science related degree. Novice programmers[2] who are enrolled in introductory programming courses are taught the fundamental concepts of computer programming and the skills to code solutions to programming problems. Success in the introductory programming course is crucial so that novice programmers are able to understand and express concepts in code, and it is a mandatory course for novice programmers who wish to obtain a qualification in the field of Computer Science and other related fields and to pursue a career as a software developer.

Reports of high failure and attrition rates in introductory programming courses are alarming. The Queensland University of Technology in Australia reported more than 30% failure rates every semester (Teague & Roe, 2009) while the University of Glasgow in Scotland reported that only 50% of their students obtained a grade C or better (Mancy & Reid, 2004). In addition, Bennedsen and Caspersen (2007) reported a 33% failure rate in their international study on introductory programming courses, and there are other reports of high failure and attrition rates in introductory programming courses (Kinnunen & Malmi, 2006; Robins, Rountree, & Rountree, 2003; Sheard & Hagan, 1998; Watson & Li, 2014).

[1] Term explained in Section 1.4
[2] Term explained in Section 1.4

The existing literature on computer programming reported that computer programming is difficult to learn, and research into the difficulties of learning computer programming go back to the 1970s (Robins et al., 2003; Soloway & Spohrer, 1989; Weinberg, 1971; Whalley & Lister, 2009). The difficulties that the novice programmers face are irrespective of the type of programming language that was used in the course; the programming environment that was used to code the solutions; and the programming paradigms that were applied to develop the solutions. As a result, novice programmers in introductory programming courses have received considerable attention from researchers and course instructors due to their inability to attain the desired performance levels (Lister et al., 2004; McCracken et al., 2001; Robins et al., 2003).

The published research on computer programming reported that several factors affect the performance of novice programmers in introductory programming courses. These range from the demands of the programming language (Guibert, Girard, & Guittet, 2004; Jenkins, 2002; White & Ploeger, 2004), the need to master multiple domains in programming (duBoulay, 1986; Samurcay, 1989), and the multiple threshold concepts in programming (Boustedt et al., 2007; Soloway & Spohrer, 1989). Other factors identified in the literature include novice programmers who appear to be using inappropriate strategies to learn programming, and had over-confident self-perceptions of their ability to learn programming (Corney, Teague, & Thomas, 2010; Lahtinen, Ala-Mutka, & Jarvinen, 2005; Milne & Rowe, 2002).

Equally, a multitude of solutions has been proposed to help novice programmers. The solutions include using technological interventions for learning programming (Blikstein, 2011; Fowler, & Cusack, 2011; Hou & Austin, 2007; Langton, Hickey, & Alterman, 2004; Lee & Ko, 2011; Murphy, Kaiser, Loveland, & Hasan, 2009; Nevalainen & Sajaniemi, 2008; Villalobos, Calderón, & Jiménez, 2009), re-structuring the pedagogical design of introductory programming courses (deBry, 2011; Ford & Venema, 2010; Moskal, Lurie, & Cooper, 2004), and identifying the predictors of success in introductory programming courses (de Raadt et al., 2005; Rountree, Rountree, & Robins, 2002).

The magnitude of the problem is further evidenced by the formation of several research groups such as the McCracken group (McCracken et al., 2001), Leeds Group (Lister et al., 2004), BRACElet project (Whalley et al., 2006; Whalley & Lister, 2009), Psychology of Programming Interest Group (PPIG) (PPIG.org), and the Swedish Group of researchers (Eckerdal et al., 2006), all of whom have been actively researching the difficulties of learning programming. Additionally, the difficulties of learning programming were also discussed during the Computer Science Curriculum review by the ACM/IEEE-CS Joint Interim Review Task Force (ACM/IEEE-CS, 2008).

Despite years of research, novice programmers continue to face difficulties when learning programming. The proposed solutions appear to be implemented or experimented within the HEI where the research was conducted, but with little evidence of these solutions being tested widely. There is also a lack of evidence of the proposed solutions being effective in minimising the high failure and attrition rates in introductory programming courses. This calls for further research on other possible factors that may help the novice programmers overcome the difficulties when learning programming. One area that has received little attention in the published literature on computer programming is the behavioural factors that affect programming performance.

1.2 Research gap

The published literature on computer programming largely attributes the poor performance of novice programmers to the demanding cognitive load of introductory programming courses (duBoulay, 1986; Jenkins, 2002; Samurcay, 1989; Stachel et al., 2013). The cognitive load is due to the nature of computer programming which is inherently complex and intellectually challenging, and may be perceived to be an external factor that affects the novice programmer's performance. Instead, Wigfield (1994) suggests that internal factors such as behaviour are within the control of the student and are a better predictor of performance. This implies that course instructors could support the novice programmer's learning by making clear behavioural expectations and designing courses which stimulate and support effective behaviour.

The published research on computer programming has examined behavioural factors such as self-efficacy, learning approaches, and self-perceptions as possible predictors of programming performance. The findings suggest that these behavioural factors have a strong influence on programming performance and that novice programmers have the ability to alter their behaviour. However, the study by Wiedenbeck, Xiaoning, and Chintakovid (2007) found that computer self-efficacy [3] affects interest and that interest in computer programming then affects the programming performance of the novice programmer. Interest is one of several the indicators of student engagement[4]. Their finding is in line with Bandura (1977) and Bandura, Adams, and Beyer's (1977) assertion that self-efficacy influences the performance of the student by influencing their behaviour.

Further, from their reading of the literature up to that time, Linnenbrink and Pintrich (2003) argued that self-efficacy affects the student's engagement in learning, and engagement then affects the performance of the students. Despite the tendency of succeeding research in the literature on learning theory to support this view, the actual relationships between self-efficacy, the indicators of student engagement, and the novice programmer's performance in learning programming have not been examined and tested further in the research on computer

[3] Self-efficacy is defined in Chapter 2, Section 2.2.1
[4] Student engagement is defined in Chapter 2, Section 2.3.1

programming. This highlights a gap in understanding the relationship between programming self-efficacy, engagement and the programming performance which this study proposes to fill.

1.3 Research Objective and Research Questions

The objective of this study is to develop and validate a model that explains the relationship between self-efficacy, engagement, and the performance of novice programmers in introductory programming courses. To achieve this, the indicators of engagement in introductory programming courses will be identified, and the effect of the novice programmer's self-efficacy belief on their engagement and the effect of their engagement on their performance in their introductory programming course will be examined.

Therefore, the research questions are:

Research Question 1 (RQ1): What is the effect of self-efficacy on the novice programmer's engagement in an introductory programming course?

Research Question 2 (RQ2): What is the effect of the novice programmer's engagement on their performance in an introductory programming course?

1.4 Terminologies

The terms novice programmer, course, higher educational institution (HEI), and programming assessments are used throughout this thesis. The terminologies are explained below:

Novice Programmer

The term novice programmer is frequently used to refer to students in introductory programming courses in the existing literature on computer programming (Lahtinen et al., 2005; Robins et al., 2003; Thuné & Eckerdal, 2009) to distinguish between the students in introductory programming courses and the students in advanced programming courses. The term novice programmer is used since admission into introductory programming courses does not require prior programming knowledge. However, exceptions may occur, as it may be possible that a few novice programmers who are enrolled in an introductory programming course may have some programming experience. This exception is proposed as a confounding variable in this study and is discussed in Chapter 3, Section 3.3.

Course

The term course is used to refer to a subject that the novice programmer is enrolled in at their HEI. A course is typically conducted over a period of one semester, follows a set of learning outcomes, and leads to assessments that contribute to the overall grade point average (GPA) of the novice programmer. HEIs may use other terms such as "module" or "unit" or "subject" to refer to a course.

Higher Educational Institution (HEI)

The term Higher Educational Institution or HEI is used to refer to universities, polytechnics, colleges, and institutes that offer tertiary education. According to the International Standard Classification of Education (2011), tertiary education "builds on secondary education, providing learning activities in specialized fields of education. It aims at learning at a high level of complexity and specialisation" and "comprises of short-cycle tertiary education, Bachelor's or equivalent level, Master's or equivalent level, and doctoral or equivalent level, respectively" (p. 48).

Programming assessment

The term programming assessment refers to the novice programmer's work that was graded during their introductory programming course. The assessments may include assignments, practical exercises, laboratory or workshop exercises, tutorial exercises, and their programming test or exam.

1.5 Structure of the Thesis

This thesis continues to Chapter 2 which examines existing literature for the three main variables in this study. They are programming and performance, self-efficacy, and student engagement. Chapter 3 then proposes the conceptual research model and the hypotheses for this study, and Chapter 4 outlines the proposed research methodology for this study. Chapter 5 then presents the analysis and findings of the focus groups, and Chapter 6 discusses the scale development for the survey questionnaires. Chapter 7 presents the analysis of the survey questionnaire, followed by Chapter 8 which discusses the overall findings of this study, and Chapter 9 which offers further insights into the findings of this study and argues the implications of the findings. Finally, Chapter 10 concludes the thesis with a summary of the research, discusses the contributions and limitations of this study, and makes recommendations for future research.

Chapter 2: Literature Review

The literature surrounding this study was systematically reviewed using a structure and strategy, and these are used to frame the literature review.

Structure

This literature review discusses the three variables in this study. They are programming performance, self-efficacy, and student engagement. The literature review begins by providing an overview of programming and examines the literature for factors that affect the performance of novice programmers. The solutions that have been proposed to overcome the high failure rates are then examined followed by a discussion on self-efficacy and student engagement. The literature review concludes by arguing the importance of examining the relationship between self-efficacy, engagement, and programming performance. This literature review examines and discusses the literature that was published prior to June 2013, which is prior to the data collection phase in this study.

Strategy

Research in the Information Systems (IS) discipline may be examined from the behavioural science paradigm or the design science paradigm (Hevner, March, Park, & Ram, 2004). Hevner et al. (2004) argued that research in the behavioural science paradigm involves the "development and verification of theories that explain or predict human organizational behaviour", while research in the design science paradigm "extends the boundaries of human and organizational capabilities by creating new and innovative artifacts" (p. 75). Existing evidence from the literature on computer programming suggests that the difficulties of learning programming have been investigated from the behavioural science and the design science paradigms. Both paradigms have made important contributions to the programming discipline. Since both research paradigms lead to different types of outcomes, this study uses the behavioural science paradigm as the aim of this study is to understand the behaviour of novice programmers, and to develop and validate a model that may predict the performance of novice programmers in introductory programming courses.

2.1 Programming

This section examines the concept of computer programming, followed by evidence of failure in introductory programming courses. Next, the challenges in the programming discipline are discussed by comparing how learning programming differs from learning in other disciplines. The factors affecting programming performance and the efforts that have been made to improve the performance of novice programmers are examined. Examples from research in the design science paradigm may be used to discuss innovative findings and solutions to overcome the difficulties of learning programming.

2.1.1 Definition of Programming

A rigorous literature search was conducted to examine the definition of programming. The search involved the examination of two key databases in the field of Computer Science, two journals in the field of Information Systems, and one general database. The Computer Science-related databases were ACM Digital Library and ProQuest Computing while the Information Systems journals include MIS Quarterly and Information Systems Research. Google Scholar was also used to search for literature on the definition of programming.

Table 2.1 lists the researchers who have discussed the concept of programming and their definition of programming. In order to ensure that the definition of programming is current, the research papers that were published before 1980 were excluded from Table 2.1. An analysis of the various definitions of programming suggests that programming is a complex problem-solving activity that requires the analysis and design of a solution to a problem and the solution is then translated into code that is understood by the computer. Although several researchers have attempted to define programming, their definitions appear to be incomplete given the growth in the programming discipline in recent years. The growth in the programming discipline may be observed by the various programming paradigms and programming languages that are used for learning programming, the increased availability of programming tools for learning programming, and the larger domain of programmers since introductory programming courses are now offered in not only Computer Science but also in other disciplines.

The approach dominating introductory programming courses has evolved from a structured programming paradigm to a modular-based paradigm, and in the 1990s, to an object-oriented based programming paradigm (Raccoon, 1997). Examples of programming languages that are based on an object-oriented programming paradigm include Java, C++, Python, and C#. The study by Nevins (2013) is recent evidence of the widespread use of the object-oriented programming paradigm in introductory programming courses. He found that in 2012, 79% of the Community Colleges in California used either Java or C++ in their introductory programming courses. In another survey, Mason, Cooper, and de Raadt (2012) found that in 2010, Java was the most popular programming language, accounting for 36.4% of programming languages used in programming courses at Australian universities.

The second evidence of growth in the programming discipline is in the increasing availability of programming tools that have simplified and eased programming tasks by minimising the need to write complex syntax. Such programming tools include libraries (Ippolito, 1997), frameworks (Powers et al., 2006), and visual programming tools (Powers et al., 2006). Additionally, a more recent technique which uses web applications, better known as “mashups” has enabled the integration of information from multiple data sources in order to build an application and does not require any programming knowledge (Tuchinda, Knoblock, & Szekely, 2011). However, although mashups do not require programming knowledge, researchers have argued that

programmers need to have an understanding of programming concepts in order to integrate information using this technique (Tuchinda et al., 2011; Zang, Rosson, & Nasser, 2008).

Table 2.1: Definition of programming

Author	What is Programming?
Jinwoo & Lerch (1997)	a scientific discovery in multiple problem spaces
Samurcay (1989, p.162)	"… producing a solution to a problem, designing a procedure to solve the problem by means of a technological tool, and writing the procedure using computer codes"
duBoulay (1986)	• Understanding programs and how problems can be approached (Orientation) • Understanding the computer based on the execution of the program (Notional machine) • Mastering the syntax and semantics of the programming language (Notations) • Notations used to form a schema or a plan to achieve a goal (Structures) • Acquiring the skills for specifying, developing, testing and debugging a program (Pragmatics)
Blackwell (2002, p. 204)	"Programming is the "spadework" of finding a precise mathematical formulation and method of solution, possibly notated in a "convenient problem-oriented language" whose symbols are "more closely related to the mathematical problem to be solved".
Pair (1993) Kelleher, & Pausch (2005)	Using computer specific codes, the programmer translates the solution from human language to computer code by writing instructions to perform a sequence of calculations for the computer to compute and execute
Eckerdal & Berglund (2005, p. 141)	… "learning to program is a way of thinking, which enables problem-solving, and which is experienced as a "method" of thinking"
ACM/IEEE-CS (2008)	Interpreted through the learning objectives of the programming fundamentals course: Programming encompasses the analysis, modification, and expansion of programs by selecting and applying appropriate constructs in order to solve a programming problem.
Ko et al. (2011)	… process of planning or writing a program

The third evidence of growth may be observed by the larger domain of programmers. Programming has extended from being a specialised skill which requires professionally trained programmers in the mathematical domain, to a broader domain of programmers including those who have little or no programming experience (more commonly known as end-users) (Blackwell, 2002; Ko et al., 2011; Wiedenbeck, 2005). These end-users acquire programming skills to perform simple programming tasks such as developing web applications, working with

spreadsheets for financial tasks, analysing business trends, and creating macros to perform calculations for their personal needs (Ko et al., 2011; Wiedenbeck, 2005). In addition, Ko et al. (2011) argued that expert programmers may engage in end-user programming in order to develop applications for their own use. As a result of the wider domain of programmers and in order for graduates to meet the skills requirement at their workplace, programming courses have extended from the Computer Science-related degrees to the business and sciences-related degrees (Wiedenbeck, 2005).

Despite these advances in the programming discipline, there does not appear to be any literature that specifically discusses how the approaches to programming have evolved over the years, and if the definition of programming should be broadened to include the recent advances in the programming discipline. However, for the purpose of this study, the interpretation of programming that was offered by ACM/IEEE-CS (2008) in Table 2.1 will be accepted as the working definition of programming in this study.

2.1.2 Evidence of failure

The difficulties of learning programming have been examined from as far back as the 1970s. These difficulties affect the performance of the novice programmer, leading to high failure and attrition rates. Failure and attrition rates appear to be more than 30% in most introductory programming courses. For example, Queensland University of Technology in Australia reported more than 30% of their students fail every semester in their Bachelor of Information Technology programme (Teague & Roe, 2009), while the University of Glasgow in Scotland reported that only 50% of their students had obtained a grade C or better between 2002 and 2003 in their Computer Science programme (Mancy & Reid, 2004). Guzdial and Soloway (2002) reported that 15 – 30% of students in introductory programming courses in the US drop out of the course or fail, while McKinney and Denton (2004) reported that between 30% and 50% of their students completed the introductory programming course at the University of South Alabama, USA. Further, Bennedsen and Caspersen's (2007) international study on 62 Higher Educational Institutions (HEIs) revealed that 33% of the students failed in their introductory programming course.

Although other researchers do not provide failure and attrition rates, the literature discussed in Sections 2.1.4 and 2.1.5 explains the difficulties of learning programming and its effect on the novice programmer's performance. In addition, anecdotal evidence suggests that the performance of novice programmers in introductory programming courses does not correlate well with performance in other academic courses (Simon et al., 2006).

2.1.3 How learning programming differs from learning in other courses?

There appears to be little discussion on how learning programming differs from learning in other courses. In a study by Zander et al. (2009), the participants explained that learning to program required an "active" learning style compared to learning mathematics which required a

"reflective" learning style. These learning styles are based on the Felder-Silverman (1988) learning style model.

On the other hand, several researchers have attempted to explain the nature of programming and why learning programming is cognitively demanding. Section 2.1.4.1 discusses the nature of the cognitive load when learning to program and suggests that programming requires mastery of multiple domains which appear to be unique to learning programming, and a novice programmer may find these domains complex and difficult to master.

Further, anecdotal evidence from discussions with introductory programming course instructors suggests that writing programming codes requires a high level of precision and the Integrated Development Environment (IDE) that is used for writing and compiling programming code has the advantage of being able to provide novice programmers with immediate feedback on their programming errors. Programming codes that are not syntactically and logically precise will result in programming errors and the program failing. The IDE is able to provide students immediate automated feedback on the nature and location of their programming errors. By contrast, many other disciplines appear to lack any plausible mechanism for effective automated feedback. However, the automated feedback provided by the IDE may not be easily comprehensible by the novice programmer, resulting in frustration and a lack of motivation to progress in the course. Chapter 5, Section 5.5 provides evidence from the focus groups participants who explained how learning programming is different from learning in other courses.

2.1.4 Factors affecting programming performance

Learning to program is cognitively demanding and can lead to poor performance in the introductory programming course. But, the behaviour of the novice programmer appears to be another factor that may lead to poor programming performance. The literature on the difficulties of learning programming appears to emphasise the cognitive demands of learning programming, and how that might affect the programming performance of the novice programmer, but places less emphasis on how the behaviour of the novice programmer might affect their programming performance.

2.1.4.1 Cognitive Load

At the core, programming is a problem-solving activity where a problem is translated into a set of instructions that is understood and executed by the computer. Chapter 1, Section 2.1, explains why the problem-solving activity is cognitively demanding. The main factors contributing to the demanding cognitive load in introductory programming courses include the demands of the programming language, the multiple domains of programming, and troublesome programming concepts.

Demands of the Programming Language

Programming languages present challenges in the form of the rigor of the programming language, the high-level of precision required, and the cognitive challenge of different programming paradigms. The programming languages used in introductory programming courses are usually driven by industry needs and are better suited for expert programmers as they contain inflexible and rigorous syntax (Jenkins, 2002; Lahtinen et al., 2005).

A high level of precision is also required when using programming languages, failing which, syntactical, semantic, and pragmatic errors may be made (Guibert et al., 2004). Pragmatic errors are made when the programming problem is not addressed completely, resulting in errors that are far more difficult to detect as compared to syntax and semantic errors (Guibert et al., 2004).

Different programming paradigms such as procedural, object-oriented and visual programming paradigms also contribute to the demanding cognitive load on novice programmers. White and Ploeger (2004) suggest that different programming paradigms present different levels of cognitive challenge, and that the programming language used should ideally strive to balance the cognitive characteristics of the novice programmer and the cognitive requirements of the programming language (White & Sivitanides, 2002).

Mastery of multiple domains

It has been argued that introductory programming courses require mastery of multiple domains which may place demands on the working memory of the novice programmer (duBoulay, 1986; Samurcay, 1989). duBoulay (1986) outlined five domains that novice programmers need to master when learning programming. The domains include orientation, notional machine, notation, structures, and pragmatics. Each domain is complex and requires multiple tasks to be accomplished simultaneously when writing a program. Table 2.2 explains each domain.

Table 2.2: Domains of difficulties in programming (duBoulay, 1986)

Domains	Description
Orientation	Understanding what is programming, how problems can be approached and the advantage of having programming skills
Notional Machine	Understanding the computer based on the execution of the program
Notation	Mastering the syntax and semantics of the programming language
Structures	Notations used to form a schema or a plan to achieve a goal
Pragmatics	Acquiring the skills for specifying, developing, testing and debugging a program

Similarly, Samurcay (1989) argued that programming requires producing a solution to the problem, writing a procedure for the solution, using a development environment to construct the procedure, and writing the codes that the computer can execute. In the same vein, Spohrer and Soloway (1989), and Winslow (1996) argued that novice programmers appeared to face difficulties in combining the algorithmic structures into programs, and lack problem-solving skills (Gomes & Mendes, 2007).

For a novice programmer, the multiple domains in programming can be overwhelming to master all at once. However, if all the domains are not mastered, the novice programmer may not be able to successfully code a solution to the problem or acquire the required programming skills. This results in the novice programmer forming poor or inappropriate mental models to solve a programming problem (Blackwell, 1996; Milne & Rowe, 2002), and poor problem-solving skills (Lister et al., 2004; McCracken et al., 2001).

Troublesome programming concepts

Troublesome programming concepts are another form of cognitive load that a novice programmer faces when they attempt to master their introductory programming course. If the novice programmer does not overcome a troublesome concept, their learning may be hindered as they may become stuck, frustrated, lose interest in the subject, adopt a surface learning approach, or even withdraw from the course (Boustedt et al., 2007; Davies, 2006; Sorva, 2010).

Meyer and Land (2003) proposed the term threshold concept to explain these troublesome concepts and defined threshold concept as "akin to a portal, opening up a new and previously inaccessible way of thinking about something. It represents a transformed way of understanding, or interpreting, or viewing something without which the learner cannot progress (p.1)."

Meyer and Land (2006) argued that threshold concepts demonstrate five characteristics. Threshold concepts are *transformative* where a learner potentially sees a transformation in the way he or she understands the subject once the learner understands a concept. The effect of this transformation may then be *irreversible* since the learner is not likely to forget the concept, and the understanding of the concept is not likely to be undone easily. Threshold concepts are *integrative* whereby the learner is able to see the interrelatedness of a concept. Threshold concepts are also *bounded* where boundaries between discipline areas may be marked and there may be concepts that must be understood before the boundary of another threshold concept is crossed. Finally, threshold concepts are likely to be very *troublesome* for learners.

Introductory programming courses contain a number of threshold concepts which novice programmers need to understand in order to progress in the course. Table 2.3 lists the threshold concepts in introductory programming courses that were identified in the literature on computer programming. These threshold concepts, in addition to being difficult to master, exhibit the bounded characteristic where a concept must be understood before another concept may be

crossed. As a result, a novice programmer may not be able to understand subsequent concepts should they face difficulties in understanding a pre-requisite concept.

Although researchers have identified several threshold concepts in introductory programming courses, the threshold concepts could differ between students (Eckerdal et al., 2006). In addition, both fundamental and advanced programming concepts may also be threshold concepts (Shinners-Kennedy, 2008). Further, what might be interpreted as a concept (Sorva, 2010; Zander et al., 2008) is another issue in the identification of threshold concepts in introductory programming courses.

Table 2.3: Threshold concepts in introductory programming courses

Threshold concepts in programming	Authors
Pointers and memory-related concepts	Boustedt et al. (2007); Jenkins (2002); Lahtinen et al. (2005); Milne & Rowe (2002)
State	Shinners-Kennedy (2008)
Loops	Lahtinen et al. (2005); Soloway & Spohrer (1989)
Variables	Soloway & Spohrer (1989); Samurcay (1989)
Arrays	Soloway & Spohrer (1989)
Recursion	Jenkins (2002); Lahtinen et al. (2005); Rountree & Rountree (2009); Soloway & Spohrer (1989);
Abstraction	Eckerdal et al. (2006); Jenkins (2002); Lahtinen et al. (2005)
Object - orientation	Boustedt et al. (2007); Eckerdal et al. (2006)

2.1.4.2 Behaviour – Research gap

In the literature on the teaching of computer programming, the cognitive demands of introductory programming courses appear to be the main focus of researchers compared to a smaller number of researchers who are concerned with the effect of the novice programmer's behaviour on their programming performance. The focus of the research into the behaviour of novice programmers has mainly focused on self-efficacy, learning approaches, and self-perception as possible predictors of programming performance. Although there has been less research on the behaviour of novice programmers in introductory programming courses, the findings suggest that behaviour has a strong influence on programming performance.

Self-efficacy

The concept of self-efficacy refers to what an individual believes he or she is able to do rather than their characteristics, personality or psychological traits, and is a stronger predictor of behaviour than outcome expectations or previous performance (Bandura, 1977; Bandura et al., 1977; Bandura, 1986; Zimmerman, 1995). The concept of self-efficacy and its role in Bandura's

Social Cognitive Theory is discussed in Section 2.2. The findings from the research on learning programming suggest that there is a relationship between the self-efficacy beliefs of novice programmers and their programming performance. Table 2.4 presents a summary of the findings of the research on novice programmers that are relevant to this research.

Table 2.4: Summary of research on self-efficacy and programming performance

Authors	Attributes Measured	Dependent variable	Findings
Askar & Davenport (2009)	1. Family background 2. Gender 3. Computer experience 4. Subject or career choice	Self-efficacy	• High correlation between computer experience and self-efficacy • Mother's use of computers had a significant effect on self-efficacy of students • Male students had higher self-efficacy beliefs than female students • Computer engineering majors had higher self-efficacy scores than other engineering majors
Chilton & Riemenschneider (2000)	Self-efficacy	1. Ability to write programs 2. Positive feedback	• Self-efficacy predicts the ability to write programs • Self-efficacy may be manipulated through positive feedback
Doubé & Lang, (2012)	Gender	1. Self-efficacy 2. Expectations of success	• Females showed significantly lower self-efficacy beliefs and expectations of success
Ramalingam et al. (2004)	1. Previous programming experience 2. Mental models	1. Self-efficacy 2. Performance - Grade	• Significant increase in self-efficacy during the course • Previous programming experience influences self-efficacy • Mental model of programming influences self-efficacy • Mental model and self-efficacy affect performance

Authors	Attributes Measured	Dependent variable	Findings
Ramalingam & Wiedenbeck (1998)	Gender	Self-efficacy	• No significant difference in self-efficacy between genders • Lower initial self-efficacy scores saw a higher increase in self-efficacy at the end of the course
Wiedenbeck (2005)	1. Previous programming experience	1. Pre-self-efficacy 2. Post-self-efficacy 3. Performance • *grade* • *debugging*	• Significant increase in self-efficacy during the semester • Previous programming experience is a strong predictor of pre-self-efficacy and predicts post-self-efficacy • High positive correlation between post-self-efficacy and performance (grade) • Low and negative correlation between pre-self-efficacy and performance (grade)
Wiedenbeck et al. (2007)	1. Software self-efficacy 2. Programming self-efficacy 3. Computer playfulness 4. Computer Interest	1. Computer Interest 2. Performance	• Software self-efficacy, programming self-efficacy, and computer playfulness influenced computer interest • Computer interest affected performance • Software self-efficacy and programming self-efficacy did **NOT** directly predict programming performance

Existing research on the self-efficacy beliefs of novice programmers appears to focus on identifying attributes that are predictors of self-efficacy beliefs. Although several attributes have been identified as potential predictors of self-efficacy, only prior experience in programming was found to predict self-efficacy (Ramalingam, LaBelle, & Wiedenbeck, 2004; Wiedenbeck, 2005). This finding may be explained by Bandura's (1977) assertion that individuals may develop their self-efficacy beliefs through performance accomplishments (Chapter 2, Section 2.2.3), whereby

the novice programmers with programming experience may have higher self-efficacy beliefs since they are able to program. Along the same vein, there was a strong correlation between using computers and self-efficacy, and novice programmers whose mothers use computers had a significant effect on their self-efficacy beliefs (Askar & Davenport, 2009).

The effect of self-efficacy on the programming performance of novice programmers was also examined and it was found that self-efficacy affects programming performance (Ramalingam et al., 2004; Wiedenbeck, 2005).

Since existing research has found that prior experience in programming predicts self-efficacy and self-efficacy then affects the programming performance of the novice programmer, it is possible that prior experience in programming may be a factor that may influence the programming performance of novice programmers in this study. Therefore, a recommendation is made in Chapter 3, Section 3.3 to examine prior programming experience as a potential confounding variable in this study.

Of more relevance to this study is the finding that self-efficacy affects behaviour, and behaviour then affects the performance of the novice programmer, instead of a direct relationship between self-efficacy and programming performance. One such study by Wiedenbeck et al. (2007) found that software self-efficacy, programming self-efficacy, and computer playfulness together affect interest, and that interest in computer programming then affects the programming performance of the novice programmer. Interest is an indicator of emotional engagement, which is discussed in Chapter 2, Section 2.3. The finding by Wiedenbeck et al. (2007) is in line with Bandura's (1986) assertion that self-efficacy is a predictor of behaviour, and behaviour then influences the performance of the novice programmer. The research by Wiedenbeck et al. (2007) presented the opportunity for this study to examine the relationship between self-efficacy, engagement and programming performance as no further research appears to have been conducted to examine this relationship in introductory programming courses.

Other self-perceptions

The novice programmer's self-perception of their ability to program is another factor that may have an effect on their programming performance. Novice programmers tend to be over-confident of their ability and underestimate the level of difficulty when learning programming, resulting in students reporting lower difficulty levels compared to their teachers (Lahtinen et al., 2005; Milne & Rowe, 2002).

By contrast, other studies propose a lack of motivation and lack of confidence as reasons for poor performance, and high attrition rates (Kinnunen & Malmi, 2006; Teague & Roe, 2008). Novice programmers who lack motivation find programming to be less enjoyable and were less confident in completing their course successfully (Teague & Roe, 2008).

When compared by gender, there appears to be a mixed set of findings in their self-perception of their ability to program. Based on Table 2.4, female novice programmers had lower self-efficacy beliefs compared to male novice programmers (Askar & Davenport, 2009, Doubé & Lang, 2012) and had lower expectations of success (Doubé & Lang, 2012). By contrast, Ramalingam and Wiedenbeck (1998) found that there were no significant differences in the self-efficacy beliefs between the male and female novice programmers.

Learning Approach

The type of approach used to learn programming appears to have an impact on the quality and depth of learning, and may also have an effect on the performance of the novice programmer. Ramsden (2003) explained that students may use a deep or surface approach to learning and argued that all students may demonstrate deep or surface learning depending on their learning task. The deep and surface learning approaches were derived from research by Fransson (1977), Marton and Säljö (1976), and Svensson (1977), and were further examined by Biggs (1987) and Ramsden (2003). A deep approach to learning has intrinsic value to the student, adds meaning to the learning task by connecting knowledge of what is known to new knowledge resulting in improved retention (Biggs, 1987; Marton & Säljö, 1976; Ramsden, 2003). By contrast, a surface approach to learning only has extrinsic value to the student. A surface learner focuses on rote learning, memorizes concepts, completes a task without giving any meaning to the task, and does not attempt to reflect on the concepts or facts and their relation to prior knowledge (Biggs, 1987; Marton & Säljö, 1976; Ramsden, 2003).

A third approach – strategic approach was suggested by Miller and Parlett (1974) and Ramsden (1979). A student who uses a strategic approach to learning is assessment-focused. The student focuses on what is required by the assessment in order to obtain the highest possible mark and allocates the amount of time required for the assessment based on the amount of effort needed to meet the requirements of the assessment. In addition, a strategic learner focuses on using study materials that are relevant to the assessment. However, strategic approach to learning did not appear to be frequently discussed in the prior literature and was deemed as an approach to studying rather than for learning (Morgan 1993).

Winslow (1996) argued that novice programmers tend to take a surface level approach by understanding programs "line by line" rather than seeing the program from a bigger perspective. A surface level approach can lead to the inaccurate application of knowledge when different problems are presented, which in turn affects the performance of the novice programmer.

In yet another research, Marton and Booth (1997) proposed four types of problem-solving approaches that a novice programmer may take to solve a programming problem and these approaches are categorised as either surface or deep approaches to learning. Table 2.5 presents a summary of the student's approach to programming and its relationship to deep and surface

learning approaches. Within each learning approach, there are two types of problem-solving approaches that a novice programmer might take. If the novice programmer uses a deep learning approach, the novice programmer understands the program as a whole and may use a structural and operational approach to solving programming problems. By contrast, if the novice programmer uses a surface learning approach, the novice programmer merely focuses on the program instead of the problem and may use an expedient and constructual approach to solving programming problems.

In recent years, researchers have reported a strong negative correlation between surface learning and performance, while a positive correlation was found between a deep approach to learning and performance (de Raadt et al., 2005; Diseth, Palleson, Brunborg, & Larson, 2010; Hughes & Peiris, 2006; Simon et al., 2006). Further, Diseth et al. (2010) found that the positive relationship between deep learning and achievement had reduced significantly when surface and strategic learning approaches were controlled while Hughes and Peiris (2006) found that the correlation between deep learning and performance was weak. Similarly, in the literature on learning theory, Miller et al. (1996) and Yip (2012) found that the different study strategies correlate with performance.

Table 2.5: Summary of student's approach in programming and its relationship to deep and surface learning approaches (Marton & Booth, 1997)

Learning approach	Approach as explained by Marton and Booth	Student's approach	Description
Surface	Opportunistic – focuses on the program	Expedient	Write programs based on similarity in nomenclature
		Constructual	Selects relevant constructs
Deep	Interpretative – meaning of the problem	Operational	Interprets what the program needs to do and operations to accomplish
		Structural	Interprets problem domain – features, constraints

2.1.5 Improving programming performance

A variety of solutions has been proposed in an attempt to overcome the difficulties of learning programming. The solutions include using a range of different technological interventions for learning programming, re-structuring the pedagogical design, and identifying the predictors of programming success. These solutions were largely intended to reduce the demanding cognitive load and ultimately improve the programming performance of the novice programmers.

2.1.5.1 Technological interventions

Collaborative technologies such as GHT (Group Homework Tool), JeCo (Jeliot Collaborative) and GS (Group Scribbles) enable novice programmers to work in groups and encourage learning in an interactive and fun environment (Nagappan et al., 2003; Teague & Roe, 2008). Collaborative technologies that use features such as editors or workspaces for writing codes or

creating diagrams collaboratively, a chat space and sharing of documents and programming code, have reported positive learning experiences through interaction and learning from peers (Hou & Austin, 2007; Langton et al., 2004; Moreno, Myller & Sutinen, 2004).

Simplified programming environments such as Alice, Scratch, and Greenfoot have also been suggested as possible solutions to improve programming performance. Alice, Scratch and Greenfoot are highly-assistive, visual, and syntax-free programming languages which have been designed to ease the cognitive demands of industry-standard programming languages on novice programmers. These languages facilitate the learning of programming concepts in a graphical and interactive manner, and help remove the complexities of the industry-standard programming languages (Utting et al., 2010). Once the fundamental concepts are mastered, learners may then move on to using industry-standard programming languages (Herbert, 2007).

However, these highly-assistive development environments appear to be better suited for novice programmers at the pre-University level or younger (Alice.org, n.d.; Resnick et al., 2009; Utting et al., 2010). Additionally, highly-assistive environments such as Alice and Scratch, it is argued, restrict the novice programmer from gaining the essential higher level programming skills required in an introductory programming course at the tertiary level (Dillon, Anderson, & Brown, 2012). As a result, there has been little uptake of these programming languages in introductory programming courses at the tertiary level.

The advances in design science research in the field of Software Engineering resulted in the introduction and development of technological interventions using learning analytics, game-based learning, and visualization technologies in an attempt to overcome the difficulties of learning programming. Table 2.6 summarises and categorises recent technological interventions for teaching and learning programming in design science research.

In summary, the researchers generally reported that using innovative approaches in their introductory programming courses had improved the novice programmers' engagement in the course and that the novice programmers reported improved learning experiences. However, these approaches are still in their experimental stage and have yet to report significant improvements in the programming performance of the novice programmer.

Table 2.6: Other innovative approaches to learning programming

Author(s)	Category of Innovation	Description
Blikstein (2011)	Learning Analytics and Educational Data Mining	Automatic generation of logs to infer patterns on how students go about programming.
Murphy et al. (2009)	Learning Analytics	Retina, a tool which works with a compiler – monitors and logs student programming activities.
Bednarik & Tukiainen (2004)	Learning analytics	Restricted Focus Viewer (RFW) – a remote eye tracker to track the visual attention – specifically the debugging behaviour and accuracy of programmers.
Chamillard (2006)	Game-based learning	Uses game creation tools to understand problem-solving. No programming required
Fowler & Cusack (2011)	Game-based learning	Used KODU Game Lab to teach programming.
Lee & Ko (2011)	Game-based learning	Introduced Gidget, a game robot debugging tool which blames itself for coding errors. The intention is to attribute programming errors on the compiler rather than the learner.
Li & Watson (2011)	Game-based learning	Blending programming learning tasks with the game construction process using tile-based Games.
Tillmann, Halleux, Xie, Gulwani, & Bishop (2013)	Game-based learning	Pex4Fun – an automated grading engine which enables the creation of virtual classrooms, customises existing courses, and publishes new learning materials.
Dahotre, Krishnamoorthy, Corley, & Scaffidi (2011)	Intelligent tutors	Interactive instructional materials that are tailored to the progress of each student.
Maleko, Hamilton, & D'Souza (2012)	Mobile learning	Mobile Social Learning Environment by enabling interaction between novices.
Villalobos et al. (2009)	Visualization technology	Interactive learning objects serve as computational tools that stimulate programming skills through an interactive graphical environment.
Nevalainen & Sajaniemi (2008)	Visualization technology	Experimented on the visual representation of an animation tool and student engagement.

2.1.5.2 Re-structure the pedagogical design

Improvements proposed to the pedagogical design of introductory programming courses include adopting a learner-centered approach when teaching and learning programming, re-designing assessment methods, and shifting the learning strategy to problem-solving in introductory programming courses. Examples of the improvements made, and the extent of their success are discussed below.

In one study that adopted a learner-centered approach to teaching and learning programming, an active learning space was created in a section of the introductory programming course by applying Kolb's concept of Learning Space for experiential learning (Kolb & Kolb, 2005). In doing so, the failure rates reduced from 42% to 14%, and increased participation (deBry, 2011). In another study, 71% of the novice programmers obtained, at least, a grade B by engaging in discussions on programming problems and wrote algorithms to solve programming problems (Deek, Kimmel, & McHugh, 1998). Other studies that proposed learner-centered activities using collaboration are discussed in Section 2.1.5.1.

Re-designing assessment methods is another initiative to improve programming performance. Newby and Nguyen (2010) experimented with using the same programming problem for each programming assessment in the course. However, the students were required to use different programming techniques so that they could focus on learning the programming concepts. There were six assessments in their course and they found that there was a significant difference in the mean of the scores in only the final three assessments, before and after the same programming problem was introduced in the course. In another study, Ford and Venema (2010) eliminated examinations in favour of formative assessments, and reported a significant reduction in failure rates, from between 30% and 40% to 18.2%.

Shifting the focus of the course to problem-solving is another attempt at improving programming performance. Moskal et al. (2004) reported an 84% retention rate and higher grades when Alice was used experimentally to help novice programmers gain a fundamental understanding of programming concepts, while Thuné and Eckerdal (2009) applied variation theory to help students understand key concepts, theories, and techniques in their introductory computing course and found that the novice programmers had an improved understanding of their programming assignment.

Further, several academics at the Queensland University of Technology in Australia made a major revision to the Bachelor of IT degree, including their introductory programming course (Corney et al., 2010). Revisions made to their introductory programming course include introducing a collaborative learning environment, changes in the assessment methods, focusing on problem-solving and design, and incorporating databases and Web development technologies into the programming course. As a result, failure rates dropped from 19% to 6%, attrition rates dropped from 19% to 6%, the number of plagiarism cases reduced, there were significant

improvements in attendance at lectures, tutorials, and practicals and novice programmers reported positive learning experiences.

2.1.5.3 Predictors of success

Yet another effort to improve the programming performance of novice programmers focused on identifying the factors or attributes that are likely to predict success. The attributes studied may be categorised into behaviour, cognition, prior programming experience, Mathematics ability, and demographics. While the attributes such as cognition, prior programming experience and the Mathematics ability of the novice programmer did not show consistency in the findings, the behavioural characteristics of the novice programmers consistently showed a strong influence on their programming performance, while the demographic information of the novice programmers did not influence their programming performance.

The behavioural factors of novice programmers such as effort and comfort level (Ventura, 2005), self-esteem (Bergin & Reilly, 2006), deep approaches to learning (de Raadt et al., 2005; Simon et al., 2006), and expectations of success (Rountree et al., 2002) showed a strong correlation with the novice programmer's programming performance.

On the other hand, the cognitive abilities of novice programmers such as visualization and spatial reasoning skills had an influence on the performance of the novice programmer (Simon et al., 2006), while abstraction ability did not have any correlation with programming success (Bennedsen & Caspersen, 2006).

The effect of the novice programmer's prior programming experience on their programming success appears to vary from one study to another. Ventura (2005) and Wilson and Shrock (2001) found that prior programming experience did not influence programming success. By contrast, Rountree et al. (2002) and Wiedenbeck (2005) found a relationship between prior programming experience and higher success rates. Similarly, the novice programmer's ability in Mathematics appears to vary in its effect on their programming success. Bergin and Reilly (2006) and Wilson and Shrock (2001) found that school mathematics proficiency influenced programming success, but Ventura (2005) found that the relationship between the Scholastic Aptitude Test (SAT) Mathematics scores and the programming success of the novice programmer had little predictive value.

Demographic attributes such as age, gender, and type of major generally did not have any significant influence on programming performance (Rountree et al., 2002; Ventura, 2005; Wilson & Shrock, 2001).

Despite limited research on the behaviour of novice programmers, the consistent findings of existing research suggest that further research is necessary to understand the effect of the novice programmer's behaviour on their programming performance. In particular, further research is necessary to examine the effect of the engagement of the novice programmer on their

programming performance since only two factors of engagement – effort and deep learning have been examined in introductory programming courses.

In addition, although the findings were not consistent, the prior programming experience and the academic ability of the novice programmer is proposed as a confounding variable in this study since it is possible that these two factors may affect the programming performance of the novice programmer. The academic ability of the novice programmer is conceptualised as intelligence in this study and the confounding variables are discussed in Chapter 3, Section 3.3.

2.1.6 Conceptualising programming performance

The term programming performance has been used widely in the literature on teaching computer programming. Although no specific definition could be found in the literature, an analysis of the attribute(s) used to measure programming performance in the literature on the predictors of programming success (Section 2.1.5.3) suggests that programming performance refers to an objective measure of how well the novice programmer has performed in their introductory programming course.

Research into the difficulties of learning programming frequently report failure rates based on the performance of the student and have objectively measured programming performance by using the final grade that the novice programmer obtained in their introductory programming course. Grades are assigned to every piece of assessment that the novice programmer completes in the introductory programming course and these grades cumulatively contribute to the final grade of the novice programmer's programming performance in the introductory programming course.

The final grade is usually measured using a percentage scale of 0 to 100%, and/or a letter grade ranging from A+ to F, whereby A+ refers to excellent performance and F refers to fail the course. This practice of measuring performance by using a percentage scale or a letter grade is usually in line with the HEIs policy of measuring performance which then contributes to the overall performance (or grade point average) of the novice programmer in the program that they are enrolled in.

In addition, the studies examining the predictors of programming success are frequently administered to a large population of novice programmers using a survey. In this regard, for the purpose of statistical analyses, programming success is assumed to be a dependent variable and the use of a programming grade allows for an objective measure of the programming performance of novice programmers.

2.2 Self-efficacy

As discussed in Section 2.1.4.2, self-efficacy was found to have a strong correlation with programming performance, but one study suggested that self-efficacy affects behaviour and that behaviour then affects the programming performance of novice programmers. This section examines the self-efficacy construct and its effect on behaviour and academic performance in the literature on learning theory. The literature review provides an overview of self-efficacy and Social Cognitive Theory, examines prior research on self-efficacy, discusses the sources of self-efficacy, and presents other constructs that are closely related to self-efficacy.

2.2.1 Overview and definition

Bandura (1977) proposed the self-efficacy construct to predict and explain behavioural change in individuals. Self-efficacy beliefs may determine how an individual copes with a difficult task, the effort invested, and their persistence in pursuing a task that is perceived to be difficult (Bandura, 1977). Self-efficacy is also a stronger predictor of behaviour compared to outcome expectancies or previous performance (Bandura, 1977; Bandura et al., 1977).

Bandura (1986) defined self-efficacy as:

> People's judgments of their capabilities to organize and execute courses of action required to attain designated types of performances. It is concerned not with the skills one has but with judgments of what one can do with whatever skills one possesses. (p. 391)

Bandura's definition of self-efficacy suggests that the individual's perception of their ability may influence how well they carry out a task or activity, and their ability then determines the degree of success in completing the task or activity. High self-efficacy in a task implies that the individual believes that they have the ability to accomplish that task successfully (Walker, Greene, & Mansell, 2006) while Schunk (1996) argued that self-efficacy influences an individual's choice and action. Schunk explained that an individual with high self-efficacy beliefs is likely to engage in tasks or activities that they feel they are competent in and are confident in accomplishing. However, an individual with low self-efficacy beliefs avoids a task that they feel they are not able to do and perceive that the given task is difficult to accomplish (Schunk, 1996). Additionally, highly self-efficacious students demonstrate a positive attitude towards learning. They tend to seek challenges, are intrinsically interested, set goals, persist in the face of challenge, adopt effective strategies to resolve the challenges, and easily recover from failures or setbacks (Bandura, 1997; Schunk, 1996).

Bandura's definition also implies that self-efficacy may differ between tasks. An individual may demonstrate high self-efficacy on one task but low self-efficacy in another task. This study proposes to use the term programming self-efficacy that refers specifically to the novice

programmers' judgment of their ability to learn programming and is defined in Chapter 3, Section 3.2.2.

2.2.2 Self-efficacy and Social Cognitive Theory

The self-efficacy construct is an important component in Bandura's Social Cognitive Theory (Bandura, 1977; Bandura, 1986). Social Cognitive Theory (SCT) is a widely accepted and empirically validated model that explains human behaviour (Compeau & Higgins, 1995). The SCT framework is based on the premise that human beings have the ability to control their thoughts, feelings, motivation, and actions (Bandura, 1986). Bandura refers to this ability as self-system, which allows human behaviour to be understood, predicted, and altered within a social context (Bandura, 1977a; Bandura, 1986). SCT explains human behaviour based on three interacting determinants which are components of a triadic reciprocality (Figure 2.1). The three interacting determinants are personal factors, behaviour, and the environment. The personal factors determinant is shaped by expectations, beliefs, goals, self-perceptions and intentions; while the environmental determinant refers to factors such as social norms, access in community and influence on others, and the behavioural determinant refers to skills such as practise and self-efficacy (Bandura, 1986; Pajares, 1997).

The triadic reciprocality depicted in Figure 2.1 describes human behaviour as an interaction between personal factors, the behaviour of an individual, and their environment (Bandura, 1986). The three interacting determinants influence each other bi-directionally (Wood & Bandura, 1989). However, the reciprocality does not always influence behaviour simultaneously or equally (Wood & Bandura, 1989). Additionally, because of the bi-directional influence between the behaviour of the individual and the environment, a person may be a product or producer of the environment (Bandura, 1986).

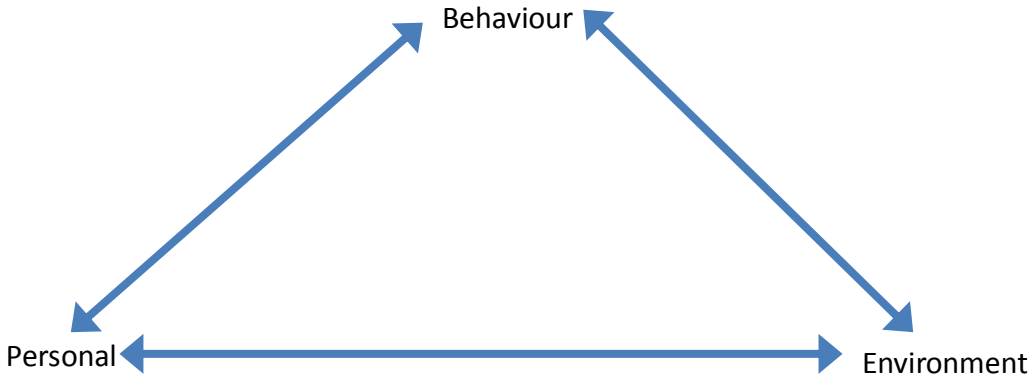


Figure 2.1: Three-way relationship in triadic reciprocality (Bandura, 1986, p. 24)

A relationship exists between each of the interacting determinants which are depicted by the arrows between the interacting determinants in Figure 2.1. An individual's thoughts, beliefs, and feelings influence how they behave. Due to the bi-directional influence between the interacting

determinants, the behaviour of the individual, in turn, influences the individual's thoughts, beliefs, and feelings. An individual's beliefs and cognitive competencies are shaped by their social interaction in the environment. The interaction with their environment then shapes the individual. Additionally, behaviour alters environmental conditions, and in return, the environment alters the behaviour of the individual (Bandura, 1986).

As an example, a programming student who encounters an error while working on a programming project might seek help from peers or a teacher (personal influences behaviour). The teacher or peer proceeds to explain or debug the error with the student (behaviour influences environment). The student then reviews the error and understands how the error was resolved (environment influences person). This suggests that the interaction between the personal factors, behaviour, and the environment influences an individual's self-belief which then affects the environment, and subsequently affects the individual's performance (Pajares, 1997).

2.2.3 Sources of self-efficacy

A closer examination of the self-efficacy construct suggests that individuals may acquire self-efficacy beliefs from four sources. They are *performance accomplishments (enactive mastery)*, *vicarious experience*, *verbal persuasion*, and *emotional arousal (physiological states)* (Bandura, 1977). These four sources are discussed below.

Performance Accomplishments (Enactive Mastery)

Performance accomplishment is highly influential in raising the self-efficacy belief of an individual and refers to an individual's success and performance in achieving a given task or situation. An individual's self-efficacy belief increases when they are successful in a given task. When the self-efficacy belief gained through performance accomplishments is strong, a subsequent failure may have little impact on the self-efficacy beliefs of the individual (Bandura, 1977; Schunk, 1991).

Vicarious Experience

Individuals form perceptions of their own ability by observing other individuals who possess similar abilities. Observing other individuals with similar abilities may convince the observer that he or she is capable of performing the same task (Bandura, 1977; Schunk, 1991). Although this source of self-efficacy belief is weaker than enactive mastery, the vicarious experience is still assumed to influence the self-efficacy of an individual who may be insecure about their own ability (Bandura, 1977). However, if the individual subsequently encounters a failure, the self-efficacy beliefs gained by the individual through vicarious experiences may diminish (Schunk, 1991).

Verbal Persuasion

Verbal persuasion is a weaker source of self-efficacy belief. Individuals may receive verbal reinforcements of their abilities. Similar to vicarious experiences, encouraging verbal reinforcements may increase the self-efficacy beliefs of the individual. However, if the individual subsequently encounters a failure, the self-efficacy beliefs gained by the individual through verbal persuasion may be diminished (Schunk, 1991).

Emotional Arousal (Physiological States)

The emotional state of an individual is another source of self-efficacy belief. Emotional states such as anxiety, stress, arousal, fatigue, and moods can influence the self-efficacy belief of an individual. These emotional states may indicate an inability to perform in a given situation thereby diminishing the self-efficacy beliefs (Schunk, 1991).

This study assumes that the acquisition of self-efficacy beliefs may come from any or all of the sources of self-efficacy discussed above, and are likely to impact the self-efficacy beliefs of the novice programmer. However, the extent of the impact of each source of self-efficacy belief on the novice programmer is beyond the scope of this study.

2.2.4 Existing research on self-efficacy

Research from several disciplines showed that high self-efficacy beliefs led to a higher ability to reach the desired performance level or accomplish a certain task. Originating in the field of psychology, self-efficacy has also been widely studied in the field of education where human behaviour plays an important role in accomplishing a given task or improving performance. Self-efficacy research in the education field has been applied in studies related to organizational behaviour, information systems, and in computer science. Some examples of self-efficacy research in these disciplines are discussed below.

In the psychology literature, self-efficacy is an important determinant for overcoming psychological disorders such as phobias (Bandura, Adams, Hardy, & Howells, 1980) and depression (Ehrenberg, Cox, & Koopman, 1991). Sherer et al. (1982) argued that individuals being treated for psychological disorders cannot be convinced to alter their behaviour. Instead, the individual must believe that he or she can perform the desired task and consequently alter their behaviour (Sherer et al., 1982).

By contrast, research in education generally examined the effect of self-efficacy on student motivation, learning, and performance (Phan, 2011; Pintrich & Schunk, 1996; Zimmerman, 2000). Educators acknowledged the importance of self-efficacy beliefs and its effect on

performance, perseverance to learn, and their ability to alter behaviour (Yip, 2012; Zimmerman, 2000). In addition, students with high self-efficacy beliefs tend to participate, work hard, persist, and are less emotional when they encounter hurdles (Bandura, 1997; Zimmerman, 2000).

In addition, educators have reported positive correlations between self-efficacy and learning achievement. An example of this may be seen in a recent research by Phan (2011) who examined the relationship between self-efficacy, learning approaches, and student achievement. The study revealed a positive correlation between self-efficacy and deep learning approaches, and that self-efficacy and deep learning together influenced the achievement of the student.

In the information systems discipline, researchers who study self-efficacy tend to examine the effect of self-efficacy on Information Technology (IT) and computer use (Agarwal, Sambamurthy, & Stair, 2000; Compeau, Higgins, & Huff, 1999; Shih, 2006). These studies also found that self-efficacy plays an important role in an individual's ability to effectively use computers and IT.

In computer science, researchers who study self-efficacy mainly examined the factors that affect the self-efficacy beliefs of the novice programmer in their introductory programming course. A small group of researchers examined the effect of the novice programmer's self-efficacy beliefs on their programming performance and found a positive relationship between the self-efficacy beliefs and the programming performance of the novice programmer. Although the findings suggest a direct relationship between self-efficacy and performance, research on student engagement on the other hand, suggests that self-efficacy is a contextual factor that affects student engagement, and student engagement then affects performance (Linnenbrink & Pintrich, 2003). This suggests a possible gap that this study seeks to fill by examining the effect of self-efficacy on engagement and the effect of engagement on the programming performance of the novice programmer. Chapter 2, Section 2.1.4.2 discusses the research conducted on self-efficacy and programming performance, and highlights the research gap.

2.2.5 Self-efficacy and other related constructs

Self-efficacy has been associated with other constructs which are also used to predict student learning and achievement. These constructs include self-concept, perceived control, outcome expectations, attributions (Schunk, 1991; Zimmerman, 2000), and self-determination (Deci & Ryan, 2000). Table 2.7 lists and explains the related constructs. With the exception of self-determination, the other constructs in Table 2.7 have proved to be less effective in predicting student learning and academic performance, while self-efficacy was found to be the strongest predictor of student learning and academic performance (Zimmerman, 2000).

In contrast to self-efficacy theory, self-determination theory is a theory of human motivation which emphasises the extent to which an individual may satisfy their innate psychological needs when pursuing an outcome (Deci & Ryan, 2000). Both the self-efficacy theory and self-

determination theory emphasise different aspects of human motivation but are similar in that they both appear to achieve a desired outcome.

Although both self-efficacy and self-determination theories appear to be equally important in predicting student achievement and learning, this study proposes to examine the effect of self-efficacy on programming performance as evidence from the existing literature on learning theory suggests that self-efficacy is a stronger predictor of performance.

Table 2.7: Other constructs used to predict student learning and achievement

Construct	Description
Self-concept (Rogers, 1951)	Self-descriptive construct of self-knowledge and self-evaluative feelings (Linnenbrink & Pintrich, 2003; Zimmerman, 2000; Schunk, 1991)
Perceived control (Rotter, 1966)	Refers to expectations of outcomes that may be controlled internally, through behaviour or externally (Zimmerman, 2000). Individuals may believe that outcomes may be dictated by an external source such as luck, chance or fate or alternatively, they may belief that outcomes may be dictated by their own actions (Schunk, 1991).
Expectations and value (Outcome expectations) (Atkinson, 1957)	An individual expects a certain outcome as a result of performing a certain behaviour, and the extent of value that the outcome holds for the individual (Schunk, 1991; Eccles, 1983)
Attributions (Heider, 1958)	An individual attributes their success or failure in achieving their desired outcome to factors such as effort, luck, ability and etc. (Schunk 1991; Weiner, 1985). These attributions may then influence future expectations of desired outcomes.
Self-determination (Deci & Ryan, 2000)	Self-determination theory concerns human motivation and personality, whereby an individual may be motivated by their innate need for competence, relatedness, and autonomy.

2.3 Student Engagement

Appleton, Christenson, Kim, and Reschly (2006) argued that although strong motivational beliefs (such as self-efficacy) are necessary for success in learning, the student may not necessarily be engaged in learning. Instead, self-efficacy is one contextual factor that affects engagement, and engagement then affects the performance of the student (Appleton et al., 2006; Linnenbrink & Pintrich, 2003). Engagement is the third variable that is discussed in this literature review.

In the following sections, an overview and definition of student engagement and its effect on student achievement are discussed. Existing models of student engagement are examined,

followed by issues with the development of the student engagement construct. Examples of student engagement models and frameworks, and findings from research on student engagement are also discussed.

2.3.1 Overview and Definition of Student Engagement

How students stay motivated and how they persist in learning have long been of interest to learning theorists. Student engagement is one construct which has recently emerged as a primary theoretical model for understanding which students are likely to fail, drop out from school, and for improving student motivation and achievement (Appleton, Christenson, & Furlong, 2008; Christenson, Reschly, & Wiley, 2012; Fredricks, Blumenfeld, & Paris, 2004). The student engagement construct is useful for examining and intervening with the gradual process of disengagement from learning (Appleton et al., 2006; Finn, 1989).

Although student engagement has been largely studied in schools, there is also much interest in understanding student engagement and its effect on student achievement and learning at the tertiary level. Therefore, the engagement construct applies to students at any level of study (Appleton et al., 2008; Christenson et al., 2012). The literature on student engagement discussed in this study will be based on research that was carried out at the tertiary level. However, in the absence of, or a lack of research on student engagement at the tertiary level, the school-based research may also be reported since this study focuses on novice programmers who may have recently left school, and are in the first year of study at the tertiary level.

The engagement construct is viewed as a multi-dimensional construct which is malleable, responsive to contextual factors such as: policies and practices in school, family influences, peers, and is alterable based on environmental changes (Appleton et al., 2006; Christenson et al., 2008; Fredricks et al., 2004). Chapter 2, Section 2.3.2 discusses recent models of student engagement that depict the contextual factors, dimensions, and outcomes of student engagement.

There is value in understanding and examining engagement as engaged students create conducive learning environments for themselves, display productive achievement behaviour, expend effort and persist when faced with difficult tasks, seek help, and self-monitor their learning activities (Schunk & Mullen, 2012). Positive outcomes such as academic success, social, and emotional learning outcomes have been observed as a result of students engaging in schools and learning, and an engaged student is less likely to drop out from school (Appleton et al., 2006; Finn & Zimmer, 2012; Fredricks et al., 2004). Further, the engagement construct has been shown to be iterative, as positive engagement improves outcomes, and an improved outcome can further strengthen engagement (Appleton et al., 2006; Fredricks et al., 2004).

The value of understanding student engagement may also be observed from the widespread use of the National Survey of Student Engagement (NSSE) (“About NSSE,” n.d.; Kuh, 2009), and

the Australasian Survey of Student Engagement (AUSSE) (Coates, 2010), which are two survey instruments for assessing general student engagement in the USA, Canada, and Australia.

Definition

Due to the wide body of research on student engagement, there appear to be three definitions of the multi-dimensional student engagement construct (Fredricks et al., 2004), and there appear to be three schools of thought on student engagement (Reschly & Christenson, 2012). Chapter 2, Section 2.3.3 offers further explanation.

Early definitions of engagement describe engagement as the student's psychological investment in learning in order to understand and master the skills and knowledge required in academia (Newmann, Wehlage, & Lamborn, 1992). On the other hand, Fredricks et al. (2004) defined engagement based on each engagement dimension, wherein cognitive engagement refers to investments in learning, behavioural engagement involves participation while emotional engagement focuses on the student's positive and negative relations to the environment. The majority of researchers agree with the definition offered by Fredricks and colleagues (Christenson et al., 2012).

More recently, having examined the definitions of student engagement offered by various researchers, Christenson et al. (2012) proposed the following definition of student engagement:

> Student engagement refers to the student's active participation in academic and co-curricular or school-related activities, and commitment to educational goals and learning. Engaged students find learning meaningful, and are invested in their learning and future. It is a multidimensional construct that consists of behavioural (including academic), cognitive, and affective subtypes. Student engagement drives learning; requires energy and effort; is affected by multiple contextual influences; and can be achieved for all learners. (pp. 816-817)

This definition encompasses all aspects of student engagement and captures the essence of student engagement as interpreted by the various researchers over the years, and will be used in this study.

2.3.2 Dimensions of student engagement

Over the years, several student engagement models and frameworks have been proposed, and there is still an on-going debate for a commonly accepted model for student engagement (Christenson et al., 2012). The more commonly discussed models include Finn's Participation-Identification Model (Finn, 1989), Linnenbrink and Pintrich's framework (Linnenbrink & Pintrich, 2003), Appleton et al.'s Student Engagement Model (Appleton et al., 2006), Kahu's engagement framework (Kahu, 2011), and Reschly and Christenson's Student Engagement Model (Reschly & Christenson, 2012). These models and frameworks vary in the dimensions of

engagement, indicators used to measure student engagement and the contextual factors that affect student engagement.

Of interest in this study are the Linnenbrink and Pintrich's (2003) framework, Appleton et al.'s (2006) Student Engagement Model, and Reschly and Christenson's (2012) Student Engagement Model. Table 2.8 lists the student engagement models and frameworks, provides a description of the engagement aspects that each model sets out to examine, and explains why the three models are of interest in this study.

Table 2.8: Description of student engagement models and decision to examine the model in this study

Engagement Model/Framework	Author	Examine in this study?	Description
Finn's Participation-Identification Model	Finn (1989)	No	Focuses only on the relationship between self-esteem and frustration as a determinant for withdrawing from school. Self-efficacy is not examined in this model.
Linnenbrink and Pintrich's framework	Linnenbrink & Pintrich (2003)	Yes	Proposes a relationship between self-efficacy, student engagement, and learning and achievement.
Student Engagement Instrument	Appleton et al. (2006)	Yes	Proposes that student engagement is influenced by contextual factors (which includes self-efficacy), and engagement then influences outcomes.
Kahu's Engagement Framework	Kahu (2011)	No	Builds upon and argues the weaknesses of previous student engagement models. Proposes that student engagement is influenced by structural and psychosocial influences (which includes self-efficacy), and engagement then influences outcomes. This model also proposes that sociocultural aspects influence the student engagement model and closely resembles that of Appleton et al.'s (2006) model.
Student Engagement Model	Reschly & Christenson (2012)	Yes	Builds upon Appleton et al.'s (2006) model. Proposes that student engagement is influenced by contextual factors (which includes self-efficacy), and engagement then influences outcomes.

In the main, there is general agreement that cognitive and behavioural engagement are two dimensions of student engagement (Appleton et al., 2006; Fredricks et al., 2004; Linnenbrink &

Pintrich, 2003; Reschly & Christenson, 2012). Evidence from existing literature also suggests that emotional engagement is a dimension of student engagement (Betts, Appleton, Reschly, Christenson, & Huebner, 2010). However, some models of engagement have used various terms such as motivational dimension (Linnenbrink & Pintrich, 2003), psychological dimension (Appleton et al., 2006), and the affect dimension (Reschly & Christenson, 2012) to describe the emotional engagement dimension. These dimensions also appear to identify similar indicators which are discussed in the following section.

2.3.2.1 Linnenbrink and Pintrich's (2003) Framework

Linnenbrink and Pintrich (2003) proposed the general framework in Figure 2.2 to examine the effect of self-efficacy beliefs on student engagement, and on learning and achievement. In comparison with the engagement models by Appleton et al. (2006) and Reschly and Christenson (2012), this model focuses on one contextual factor – self-efficacy as an influencer of student engagement. Linnenbrink and Pintrich's (2003) framework is particularly relevant in this study since their framework closely explains the research by Wiedenbeck et al. (2007) that found a relationship between self-efficacy, interest and programming performance (Chapter 2, Section 2.1.4.2), leading to the need for this study.

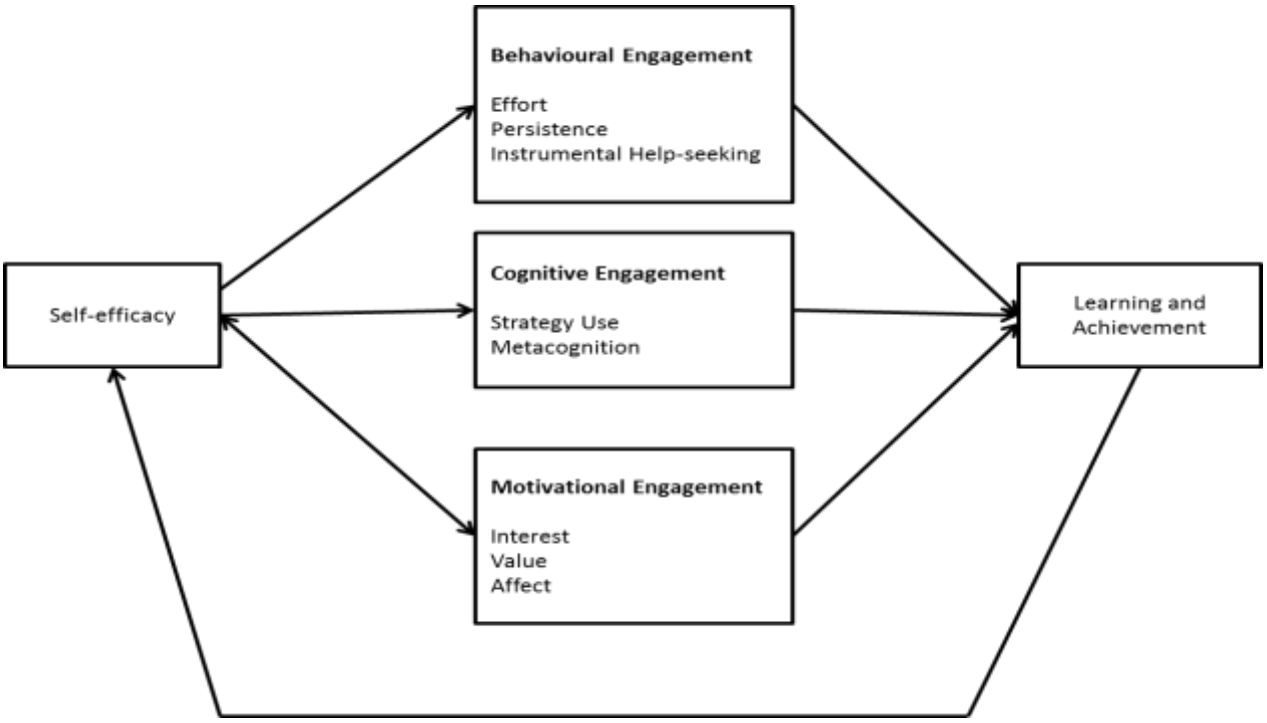


Figure 2.2: General framework for self-efficacy, engagement, and learning (Linnenbrink & Pintrich, 2003)

Linnenbrink and Pintrich's (2003) framework proposes that student engagement consists of three dimensions: *behavioural engagement*, *cognitive engagement*, and *motivational engagement*.

Linnenbrink and Pintrich (2003) argue that students must be behaviourally, cognitively and motivationally engaged in learning simultaneously. This argument is consistent across all student engagement models.

According to Linnenbrink and Pintrich (2003), *behavioural engagement* refers to observable behaviour such as effort, persistence, and instrumental help-seeking. The self-efficacy beliefs can be examined by the extent of the effort and persistence of the student in learning or achieving a task.

Cognitive engagement refers to the student's thought processes such as the strategies students use to learn, and metacognition where a student "reflects on his own thinking, actions, and behaviour and monitors and regulates their learning" (Linnenbrink & Pintrich, 2003, p. 125).

Motivational engagement or also known as *emotional engagement* (Betts et al., 2010), refers to the interest students have in the task or content they are engaged in. Additionally, the task or content the student is engaged in should be a positive experience (affect) and should be important and useful (value) for the student to learn. Thus, Linnenbrink and Pintrich (2003) proposed interest, value, and affect as measures of motivational engagement. An iterative relationship is proposed between self-efficacy and the motivational engagement construct as self-efficacy beliefs influence motivational engagement, and self-efficacy is also seen as a motivational construct (Linnenbrink & Pintrich, 2003; Pintrich & Schunk, 1996).

Linnenbrink and Pintrich (2003) also propose a feedback loop between the learning and achievement construct, and the self-efficacy construct, as high self-efficacy beliefs can improve student engagement, and subsequently improve learning and achievement. At the same time, an improvement in learning and achievement can further strengthen the self-efficacy beliefs of the student.

The existing literature on learning theory has supported the Linnenbrink and Pintrich's (2003) framework of a relationship between self-efficacy, engagement, and student learning and achievement. Chapter 2, Section 2.4 discusses the studies. Additionally, this framework has also been widely cited.

2.3.2.2 Appleton et al. (2006) Student Engagement Model

Figure 2.3 illustrates the student engagement model proposed by Appleton et al. (2006). This theoretical model underlies the development and validation of the Student Engagement Instrument (SEI) for their Check & Connect student engagement intervention (Reschly & Christenson, 2012). The SEI was developed to mainly measure the cognitive and psychological/affective dimensions of student engagement (Appleton et al., 2006).

Consistent with previous literature, the behavioural and cognitive dimension of student engagement is proposed in this model. However, Appleton et al. (2006) proposed an academic

and psychological dimension to student engagement. According to Appleton et al. (2006), the academic and behavioural dimension of student engagement consists of indicators which are observable, while the cognitive and psychological (also known as *emotional engagement* (Betts et al., 2010)) dimensions of student engagement consist of internal indicators which are less observable. Unlike the other two models reviewed in this section, this model does not propose an iterative cycle between the outcomes and engagement or a feedback loop between engagement and the context.

When compared to Linnenbrink and Pintrich's (2003) model, this model highlights several contextual factors that could influence student engagement. In this model, self-efficacy is categorised as academic beliefs and efforts and is categorised under the contextual factors involving peers.

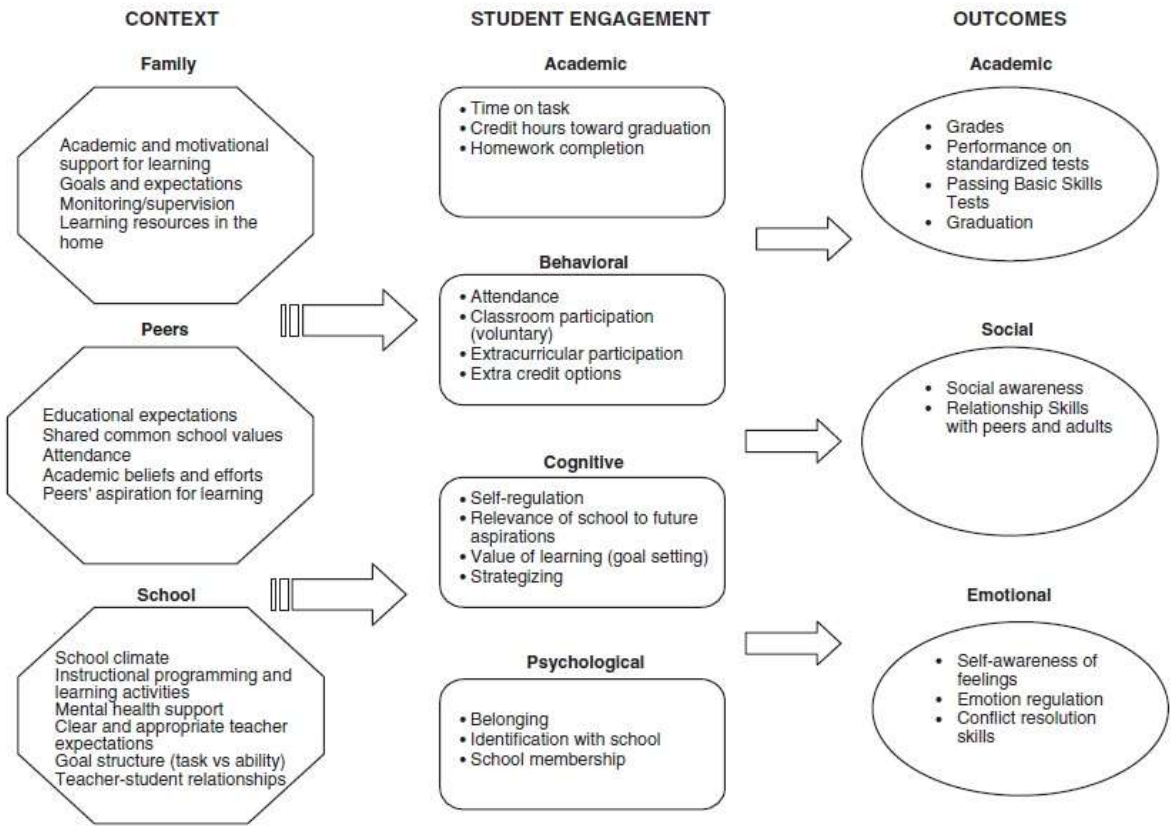


Figure 2.3: Indicators and outcomes of engagement and dimensions of engagement (Appleton et al. 2006)

2.3.2.3 Reschly and Christenson's (2012) Student Engagement Model

Like Appleton et al.'s (2006) model, Reschly and Christenson's (2012) model (Figure 2.4) was proposed as a result of their involvement in the Check & Connect student engagement intervention (Reschly & Christenson, 2012). This model builds upon Appleton et al.'s (2006) model but community is proposed as an additional contextual factor influencing student engagement. While Appleton et al.'s (2006) model depicts a linear relationship between context and the indicators of student engagement and between student engagement and outcomes, Reschly and Christenson (2012) propose an iterative relationship between the contextual factors and the indicators of student engagement.

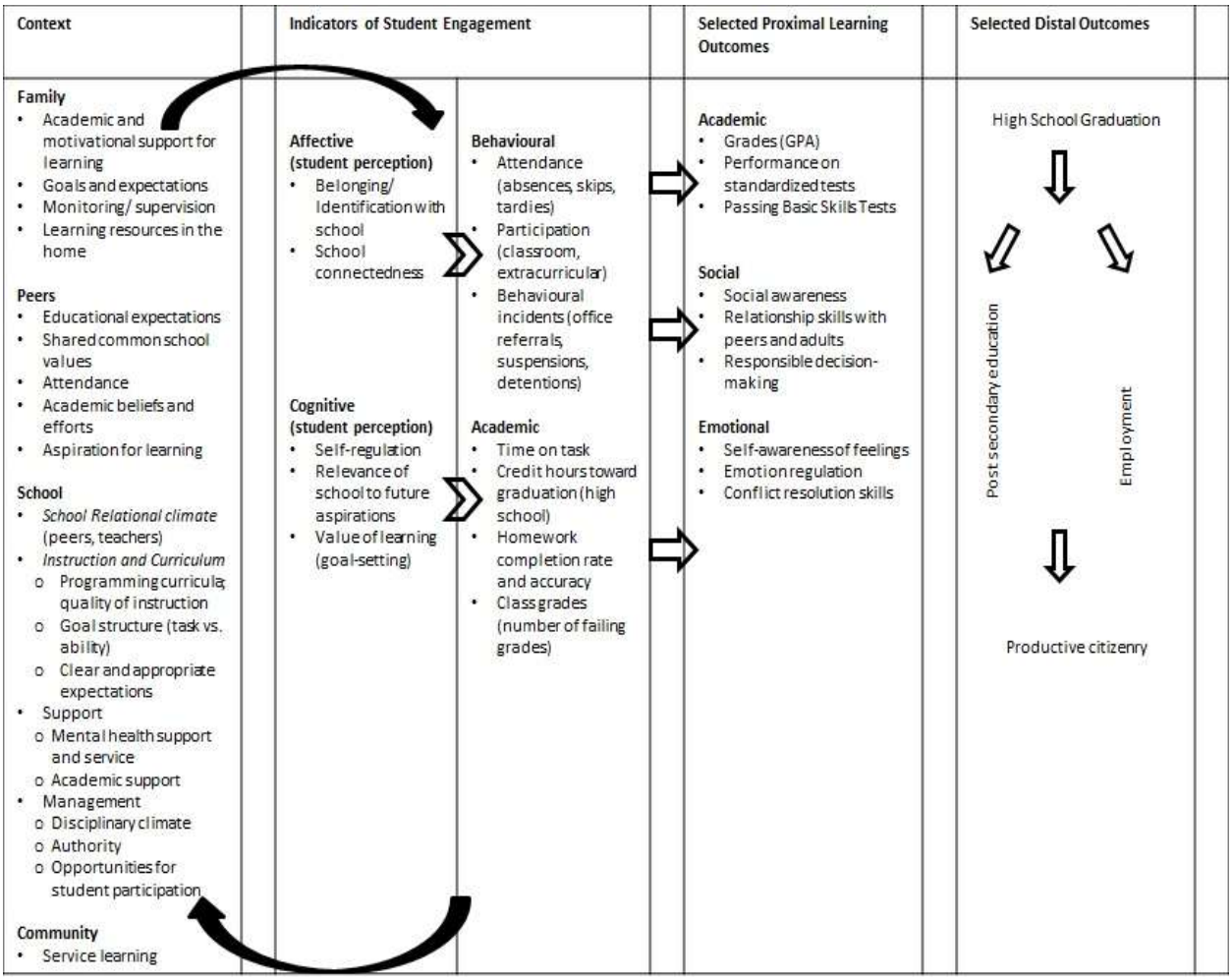


Figure 2.4: Models of associations between context, engagement, and student outcomes (Reschly & Christenson, 2012)

Additionally, Reschly and Christenson believe that student perception (affective and cognitive indicators) is at the core of student engagement and determines the student's behaviour and

learning. Therefore, they propose that affective and cognitive dimensions should precede the behavioural and academic dimensions of student engagement. The indicators for each of the dimensions are similar to the model proposed by Appleton et al. (2006). Further, like Appleton et al.'s (2006) model, this model also highlights a range of contextual factors that could influence student engagement. The model also categorises self-efficacy as academic beliefs and efforts and is categorised under contextual factors involving peers.

2.3.3 Issues with student engagement

Fredricks et al. (2004) argued that the large body of research on the various dimensions of student engagement has resulted in a lack of conceptual clarity of the construct while Reschly and Christenson (2012) argued that there are three schools of thought that make up the literature on student engagement. One school of thought arose as a result of dropout prevention theory and intervention, the second from general school reform, and the third school of thought arose from motivational literature. Further, they also argued that within these schools of thought, sub-disciplines are formed with interests in various aspects of engagement. As an example, educational psychologists are interested in examining engagement within an educational context while development psychologists are interested in engagement from the perspective of motivational theory (Reschly & Christenson, 2012). The issues are identified and discussed below.

Indicators and Facilitators

Several researchers in the field suggest that the engagement construct lacks clear distinction between the indicators and the facilitators of engagement (Appleton et al., 2008; Finn & Zimmer, 2012; Kahu, 2011; Reschly & Christenson, 2012). According to Skinner, Furrer, Marchand, and Kindermann (2008), indicators are the features that describe the engagement construct, while facilitators are the causal factors that influence engagement. Kahu (2011) argues that differing perspectives such as behavioural, psychological, socio-cultural and holistic has resulted in different models of student engagement, while Finn and Zimmer (2012) argue that some measurement scales have included context and outcomes of engagement that do not fit within the concept, causing a lack of clarity of the construct.

To resolve this dilemma, Skinner et al. (2008) and Reschly and Christenson (2012) suggest that understanding the types of research participants (for example, student, parent or teacher) may help to distinguish and maintain independence between the indicators and facilitators of engagement, while Finn and Zimmer (2012) proposed adopting the context and outcomes of student engagement that was outlined by Fredricks et al. (2004). However, Appleton et al. (2008) argued that the indicators of engagement are reflective of each engagement dimension, while facilitators make up the contextual factors that influence engagement, and propose that facilitators and indicators may be determined based on the context and duration of the study.

Process or Outcome

There is still confusion in determining whether the engagement construct is a process or an outcome. Kahu (2011) suggested that engagement can be a process when the factors that influence engagement are discussed while engagement is an outcome when referring to the psychological states of the student such as affect, cognition, and behaviour. Appleton et al. (2006) proposed that engagement is a mediator between the contextual factors and the outcomes of student engagement. This suggests that the engagement construct is viewed as a process. By contrast, Christenson and Reschly (2012) proposed engagement as either a process or an outcome depending on how researchers conceptualize student engagement in their research. As an example, engagement may be a process and an outcome if the intent is to improve the indicators of engagement in the long-term.

Dimensions

The number of engagement dimensions has also been a debate among researchers. While cognitive engagement and behavioural engagement appear to be widely accepted as dimensions of student engagement, other researchers have also proposed additional dimensions. Some examples include academic (Appleton et al. 2006; Reschly & Christenson, 2012), psychological (Appleton et al. 2006), motivational (Linnenbrink & Pintrich, 2003), and affect (Kahu, 2011; Reschly & Christenson, 2012). Section 2.3.2 discusses the engagement dimensions and their sub-types.

Engagement and Motivation

Three differing views persist on the relationship between engagement and motivation. One view claims that cognitive engagement subsumes, or is similar to, or is an outcome of motivation (Fredricks et al., 2004; Reschly & Christenson, 2012; Walker et al., 2006), but other researchers make clear distinctions between motivation and engagement (Christenson, Appleton, & Furlong, 2008; Reschly & Wiley, 2012; Schunk & Mullen, 2012), or have used both terms synonymously in their work on engagement in schools (National Research Council and Institute of Medicine, 2004).

Generally, most researchers attempted to make clear distinctions between the two constructs. Motivation refers to the process where one's energy is directed and sustained on a goal-oriented activity (Schunk, Pintrich, & Meece, 2008) while engagement is a result of motivation – where cognition, behaviour, and affect are energized, directed, and sustained on an activity (Skinner et al., 2009). Therefore, engagement and motivational constructs are separate but are related. Motivation can exist without engagement (Furrer & Skinner, 2003) and is necessary but not sufficient for engagement (Appleton et al., 2006; Appleton et al., 2008; Reschly & Christenson, 2012).

2.3.4 Student Engagement Instruments

Table 2.9 highlights six types of student engagement instruments and the indicators used to measure student engagement. The selection of the student engagement instruments was based on the extensive research by Fredricks and McColskey (2012) who examined the methods and instruments used to measure student engagement in schools. Although Fredricks and McColskey identified 11 instruments, only 6 instruments were selected for discussion in this study as they closely relate to the indicators in the three student engagement models discussed in Section 2.3.2.

Table 2.9: Self-reporting student engagement instruments (adapted from Fredricks & McColskey, 2012)

Author	Measures	Indicators
Miller et al. (1996)	Attitudes Toward Mathematics Survey (ATM)	Self-regulation Deep cognitive strategy use Shallow cognitive strategy use Persistence Effort
Skinner, Kindermann, & Furrer (2009)	Engagement vs. Disaffection with Learning (EvsD) – student report only	Behavioural engagement Behavioural disaffection Emotional engagement Emotional disaffection
www.indiana.edu/~ceep/hssse	High School Survey of Student Engagement (HSSSE)	Cognitive/intellectual/academic engagement Social/behavioural/participatory engagement Emotional engagement
Pintrich & DeGroot (1990)	Motivated Strategy and Learning Use Questionnaire (MSLQ)	Self-regulation Cognitive strategy use
Fredricks, Blumenfeld, Friedel, & Paris (2005)	School Engagement Measure (SEM)	Behavioural engagement Cognitive engagement Emotional engagement
Appleton et al. (2006)	Student Engagement Instrument (SEI)	Affective engagement: teacher-student relationships Affective engagement: peer support for learning Affective engagement: family support for learning Cognitive engagement: control and relevance of schoolwork Cognitive engagement: future aspirations and goals

Additionally, the selection of the student engagement instruments was based on the more common method of using self-reporting surveys to gather student engagement data (Fredricks & McColskey, 2012). The instruments may also contain indicators other than engagement but these

were not presented in Table 2.9 as they were not relevant to this study. These instruments have also been administered to large sample sizes, ranging from 173 students (using MSLQ) to 7200 students (using HSSE) in schools.

Fredricks and McColskey (2012) identified several differences between the instruments. Firstly, the instruments differ in terms of their focus. The ATM, EvsD and MSLQ instruments focus on class level engagement while the SEI, SEM, and HSSE instruments focus on general engagement in schools. Secondly, the engagement instruments also differ in the way they are conceptualized. Skinner et al. (2009) measure engagement and disengagement while the other instruments interpret a lack of engagement with a low engagement score.

Thirdly, these instruments also differ in the extent of their multidimensionality (Fredricks & McColskey, 2012). The HSSE and SEM instruments measure the behavioural, emotional and cognitive dimensions, the ATM, EvsD and SEI instruments measure the behavioural and cognitive dimensions while the MSLQ instrument only measures the cognitive dimension.

2.4 Self-efficacy, engagement and programming performance

This section argues the importance of examining the relationship between self-efficacy, engagement and programming performance by drawing upon the evidence presented in the existing literature on learning theory and computer programming.

2.4.1 Evidence of the relationship between self-efficacy and engagement

As discussed in Section 2.1.4.2, research examining the self-efficacy of novice programmers reported a strong direct relationship between their self-efficacy beliefs and their programming performance (Ramalingam et al., 2004; Wiedenbeck, 2005). However, their findings did not discuss how self-efficacy influences the performance of the student by influencing their behaviour. On the other hand, Walker et al. (2006) examined the relationship between self-efficacy and the cognitive engagement of 191 University students. They found that self-efficacy was a predictor of cognitive engagement. Their findings provide support for a possible relationship between self-efficacy and engagement in this study although their study was not specific to introductory programming courses. However, their study did not examine the effect of cognitive engagement on the performance of the students, and their study did not measure specific indicators of cognitive engagement and instead assumed that cognitive engagement is a high-level construct.

2.4.2 Evidence of the relationship between engagement and performance

To date, with the exception of the research by Wiedenbeck et al. (2007), there is a gap in the research on learning programming that examines the effect of student engagement on programming performance in introductory programming courses. However, there appears to be some anecdotal evidence in the research on learning programming that suggests that engagement is likely to have a strong influence on programming performance. Examples of these include:

poor attendance to tutorials and practicals (Corney et al., 2010), disengaging from programming tasks when programming errors could not be resolved (Rodrigo et al., 2009), losing interest when unable to overcome threshold concepts (Boustedt et al., 2007; Davies, 2006; Sorva, 2010), and another study that identified effort (an indicator of behavioural engagement) as the strongest predictor of programming success (Ventura, 2005).

In the education discipline, Carini, Kuh, and Klein (2006) examined the relationship between engagement and academic performance in a survey of 1058 students from 14 colleges and universities. One finding of their study revealed that the relationship between student engagement and academic performance such as critical thinking and grades was weak, but these nevertheless appeared to be positively linked. In addition, Skinner and Pitzer (2012) argued that engagement is "a robust predictor of student learning, grades, achievement test scores, retention, and graduation" (p. 21).

2.4.3 Evidence of the relationship between self-efficacy, engagement and performance

In the literature on the teaching of computer programming, only one finding by Wiedenbeck et al. (2007) suggests a possible relationship between self-efficacy, engagement, and programming performance. The study by Wiedenbeck et al. (2007) revealed that self-efficacy affects interest in programming and that interest then affects the programming performance of the novice programmer. Interest is an indicator of motivational (or emotional) engagement in the student engagement framework proposed by Linnenbrink and Pintrich (2003).

In the broader education discipline, Bresó, Schaufeli, and Salanova (2011) experimented on a cognitive behaviour-based intervention to decrease anxiety in exams. Participants in the study were at various levels of their University education. Results of the study revealed that the cognitive behaviour-based intervention group had increased self-efficacy, engagement, and performance. In another investigation, Galyon, Blondin, Yaw, Nalls, and Williams (2012) found that academic self-efficacy predicted class participation and exam performance in a survey involving 165 undergraduates in a human development course. Participation is an indicator of behavioural engagement.

To further argue the need for this study, learning theorists suggested that self-efficacy affects engagement and learning (Pintrich & Schunk, 1996; Schunk & Mullen, 2012). This is because students engaged in learning are also self-efficacious learners who strive to fulfill their outcome expectations, value learning, are goal-oriented, stay focused, and use learning strategies that they believe will lead to success (Schunk & Mullen, 2012). In addition, the literature on learning theory suggested that self-efficacy is one contextual factor that influences how a student engages in a course while academic performance is one outcome of engagement (Bresó et al., 2011; Linnenbrink & Pintrich, 2003; Schunk & Mullen, 2012). This evidence in the literature on learning theory and the literature on computer programming suggests a possible relationship

between self-efficacy and engagement, and between engagement and programming performance which leads to the need for this study.

2.5 Chapter Summary

The cognitive load in introductory programming courses and the behaviour of novice programmers are two main factors that affect their programming performance and these are discussed in Section 2.1.4. The novice programmer has little control over the content that is delivered in their introductory programming course but their behaviour is one factor that they are able to control in order to manage the demanding cognitive load of their introductory programming course. This suggests that course instructors could support the novice programmer's learning by making clear behavioural expectations and by designing courses which stimulate and support effective behaviour.

While a number of research in the literature on teaching computer programming have examined the cognitive load in introductory programming courses (Section 2.1.4.1), and solutions have been proposed to minimise the cognitive load (Section 2.1.5), the existing research on the behaviour of novice programmers in introductory programming courses appear to be limited although behaviour has a strong influence on the programming performance of the novice programmer (Section 2.1.4.2). Self-efficacy and engagement are two behavioural factors that emerged from the published literature on learning theory and computer programming and were found to have an impact on the performance of the students. While the direct relationship between self-efficacy and programming performance has received considerable attention, there is little evidence in the literature on the teaching of computer programming that examines the relationship between self-efficacy, engagement, and programming performance. This observation suggests a gap in the published research on the teaching of computer programming which this study attempts to fill.

The literature on self-efficacy and Social Cognitive Theory leaves little doubt that examining an individual's self-efficacy can provide useful insights on the behaviour of an individual in introductory programming courses. In addition, there is strong evidence in the existing literature on learning theory that argues the value of understanding student engagement despite issues raised on the lack of clarity in conceptualising the engagement construct, and the lack of standardised measurement scales. Finally, the evidence shown in the literature on learning theory and computer programming highlights the importance of examining the relationship between self-efficacy, engagement and the programming performance of novice programmers in introductory programming courses.

Chapter 3: Conceptual Research Model and Research Hypotheses

The conceptual research model that will be used to examine the research questions in this study is developed by presenting an initial research model for the three key constructs in this study. Each key construct in the model is then discussed individually and hypotheses are proposed. In the discussion on the engagement construct, the higher-order engagement constructs are proposed, potential indicators of the engagement constructs are reviewed, and hypotheses are proposed. The confounding variables that may affect the dependent variable in this study and the formative and reflective constructs in this study are proposed and discussed, followed by the full research model with the hypotheses.

3.1 Initial Research Model

Figure 3.1 presents an initial research model depicting the three constructs in this study and their proposed relationships. This model is a minor refinement of the Linnenbrink and Pintrich's (2003) framework (Chapter 2, Section 2.3.2.1) and from the review of the dimensions of student engagement in Chapter 2, Section 2.3.2. The model illustrated in Figure 3.1 presents the three key constructs in this research. They are *programming self-efficacy, engagement,* and *programming performance*. The relationships between the constructs are depicted by the arrows connecting the constructs. The research model also shows the multi-dimensional engagement construct which is made up of *behavioural engagement*, *cognitive engagement,* and *emotional engagement*. Behavioural engagement, cognitive engagement, and emotional engagement were selected as the three dimensions of student engagement in this research since these three dimensions are commonly used in the research on student engagement (Fredricks et al., 2004). This study seeks to answer Research Question 1 (RQ1) by proposing a relationship between programming self-efficacy and the engagement of the novice programmer, and seeks to answer Research Question 2 (RQ2) by proposing a relationship between engagement and the programming performance of the novice programmer an introductory programming course.

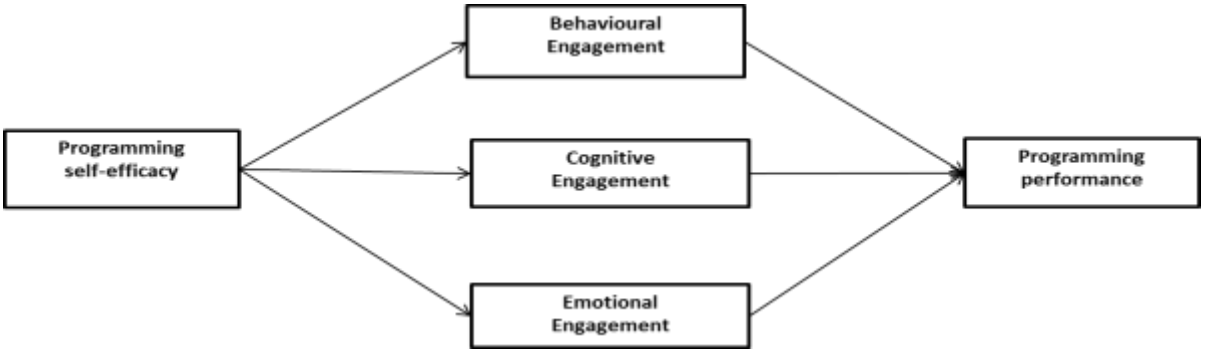


Figure 3.1: Initial research model

This model does not propose a feedback loop between programming performance and self-efficacy as this would require a longitudinal study of the self-efficacy beliefs of the novice programmers in intermediate or advanced programming courses (upon completion of the introductory programming course), and is not within the scope of this study.

3.2 Constructs and Research Hypotheses

The constructs presented in the initial research model in Figure 3.1 are developed and discussed in this section and the research model, the relationships, and the hypotheses proposed in this study are depicted in Figure 3.2 in Section 3.5. The proposed indicators for each construct and their corresponding hypotheses are discussed. Evidence of research supporting the development of the indicators for each construct will be derived from the literature on computer programming and from the literature on learning theory.

3.2.1 Programming Performance: the dependent variable

In this study, the purpose of the dependent variable programming performance is to objectively measure how well the novice programmer has performed in their introductory programming course and is discussed in Chapter 2, Section 2.1.6. Programming performance is then defined as the novice programmer's level of success in attaining the outcomes of their introductory programming course. The existing literature on computer programming frequently uses course grade (deBry, 2011; Ford & Venema, 2010; Moskal et al., 2004), and to a lesser extent self-assessment (Rajkumar, Anderson, Benamati, & Merhout, 2011) to measure the novice programmer's performance in introductory programming courses. Thus, this study proposes *programming grade* and *self-assessment* as measures of programming performance.

Programming grade

Although course grades are frequently used as a measure of programming performance, some researchers argue that course grades may not accurately reflect the programming ability of the novice programmer. This limitation was observed in Ford and Venema's (2010) study where examinations were eliminated in favour of formative assessments. Subsequent diagnostic tests revealed that novice programmers had a poor grasp of programming knowledge leading to concerns that the change in the assessment method had over-inflated the grades (Ford & Venema, 2010).

Further, during a discussion with a senior academic, concerns were raised about the possibility that the course grades might also assess the student's participation or other non-programming related tasks, and that these may not accurately reflect programming performance. In this regard, participation is an indicator of engagement and may be considered as a measure of programming performance, while other assessments that involve non-programming related tasks will be considered as an indicator of programming performance if the assessment addresses the learning outcomes of the introductory programming course.

In addition, to limit the possibility of a non-programming related assessment, the course outlines will be examined to ensure that the course grades measure the learning outcomes of the introductory programming course. From a statistical perspective, course grade is a single-item measure for programming performance. Diamantopoulos, Sarstedt, Fuchs, Wilczynski, and Kaiser (2012) advised researchers to thoroughly consider the use of single-item measures since the predictive validity of multi-item scales tend to outperform single-item scales (p. 434).

On the other hand, the student engagement models discussed in Chapter 2, Section 2.3.2 have identified several ways to assess performance, which these models refer to as outcomes. Linnenbrink and Pintrich (2003) proposed learning and achievement as an outcome of student engagement while Appleton et al. (2006), and Reschly and Christenson (2012) proposed that the outcome of student engagement may be assessed based on the student's academic achievement, social achievement, and emotional achievement. Although these student engagement models proposed three broad areas of assessing the outcomes of engagement, existing research on university-wide student engagement tends to assess the student's performance based on course grades (Fuller, Wilson, & Tobin, 2010; Kuh et al., 2008). The course grade is the preferred measure of success as these studies were carried out in Higher Educational Institutions (HEIs) where grades are strong indicators of success and enable progression to the next level of study. As such, based on evidence from existing research, this study proposes to use the final introductory programming course grades as an indicator of programming performance.

Self-assessment

To address Diamantopoulos et al.'s (2012) concern over the use of a single-item scale to measure programming grade, this study also proposes *self-assessment* as an indicator of programming performance and is measured using a multi-item scale. Mills, Pajares, and Herron (2007) define self-assessment as "the individual's assessment of their abilities after they have completed a particular activity or task" (p. 421). This indicator requires the student to self-assess their ability to program at the end of their introductory programming course.

A study conducted by Rajkumar et al. (2011) in an Information Systems programme found that a student's self-assessment of their performance is a valid proxy to assess learning outcomes that are related to the student's technical ability. This finding is consistent with Falchikov and Boud's (1989) finding that student self-assessments in science courses are more accurate compared with self-assessments in social science courses as science courses tend to focus on correct answers while social science courses require understanding and application of concepts. Therefore, due to the high level of precision required in writing a computer program, and the fact that introductory programming courses belong to the field of sciences, student self-assessment is proposed as a multi-item construct to measure programming performance.

The novice programmer's self-assessment of their performance can then be compared with their actual performance (programming grade) in the introductory programming course. The hypotheses for the dependent variables are discussed in Section 3.2.3.

3.2.2 Programming self-efficacy

Self-efficacy refers to what an individual believes he or she is able to do rather than their characteristics, personality, or psychological traits (Bandura, 1977; Zimmerman, 1995). Lorsbach and Jinks (1999) describe self-efficacy as "a sense of confidence regarding the performance of specific tasks" (p. 157). Using these two definitions of self-efficacy, this study defines programming self-efficacy as the novice programmer's judgment of their ability to learn programming.

This study measures the programming self-efficacy beliefs of novice programmers by measuring the change in their self-efficacy beliefs at two intervals. This is because Ramalingam et al. (2004) and Wiedenbeck (2005) found that the self-efficacy beliefs of novice programmers increased as a result of instruction and having hands-on programming experience during the course.

Two indicators are proposed to measure the change in the programming self-efficacy beliefs of the novice programmers. The first indicator, *pre-programming self-efficacy*, measures the self-efficacy beliefs of the novice programmer at the beginning of the introductory programming course so that the novice programmer's belief in their ability to learn programming may be established. Pre-programming self-efficacy is defined as the belief of the novice programmer in performing programming related tasks at the beginning of their introductory programming course. The second indicator, *post-programming self-efficacy*, measures the self-efficacy beliefs of the novice programmer at the end of their introductory programming course and is defined as the belief of the novice programmer in performing programming related tasks at the end of their introductory programming course. The following hypothesis is then proposed:

H1: Pre-programming self-efficacy beliefs will have a positive effect on post-programming self-efficacy belief.

3.2.3 Engagement

This study examines the *behavioural*, *cognitive* and *emotional* dimensions of engagement that were proposed by Fredricks et al. (2004) as the literature on student engagement has clearly argued the importance of these three engagement dimensions for learning and academic performance (Chapter 2, Section 2.3).

However, for each dimension, the potential indicators of engagement in introductory programming courses were identified by drawing upon evidence from the existing literature on computer programming, and from the existing literature on learning theory.

Since the engagement construct in this study is a multi-dimensional construct, the hypotheses are then proposed for the three dimensions of engagement and are referred to as the hypotheses for the higher-order constructs, and the hypotheses for the indicators of engagement within each dimension are referred to as the hypothesis for the lower-order constructs. The concept of higher-order and lower-order constructs is discussed in Section 3.4.

3.2.3.1 Review of Indicators for Engagement

Table AK.1 in Appendix K lists the indicators that were considered for each engagement dimension by listing the researchers that have examined the relationship between self-efficacy and the proposed indicator of engagement, or the relationship between the proposed indicator of engagement and performance. These informing references are from research in the programming discipline, or from other disciplines. The findings of these studies are also discussed in Table AK.1, and a decision column has been included to identify whether the indicator should be accepted, rejected, or is likely to be an indicator of engagement in an introductory programming course.

The strategy used to identify the proposed indicators in Table AK.1 was by initially including the indicators from the engagement models in Chapter 2, Section 2.3.2. Next, a brainstorming session with two academics who were involved in teaching programming in the School of Information Management, Victoria University of Wellington, identified other likely indicators of student engagement in programming. Finally, the proposed indicators of engagement were also identified through a literature search in the ACM Digital Library and the ProQuest Computing databases.

The following are the criteria used to decide if a construct should be accepted, rejected, or is likely to be an indicator of engagement in a programming course:

1. Evidence from the programming discipline and/or other disciplines that have examined the relationship between the suggested indicator and performance; and
2. If the indicator is likely to be an indicator of engagement in a programming course based on the definition of engagement discussed in Chapter 2, Section 2.3.1.

Based on Table AK.1, this study proposes *participation, help-seeking, persistence and effort* as indicators of behavioural engagement; *deep learning, surface learning,* and *trial and error* as indicators of cognitive engagement; and *interest* and *enjoyment* as indicators of emotional engagement.

3.2.3.2 Behavioural Engagement

Behavioural engagement refers to the student's involvement in observable academic, social and extracurricular activities (Fredricks et al., 2004; Yazzie-Mintz, & McCormick, 2012). This study will focus on only the observable academic activities of the novice programmers. The observable social and extracurricular activities will be excluded from this study as they are beyond the scope of this research.

The following higher-order hypotheses for behavioural engagement are proposed:

HPSEB: Self-efficacy beliefs in learning programming will have a positive effect on behavioural engagement.

HBPG: Behavioural engagement in learning programming will have a positive effect on course grade.

HBSA: Behavioural engagement in learning programming will have a positive effect on self-assessment.

The indicators proposed for behavioural engagement are *participation, help-seeking, effort,* and *persistence*.

Participation

Participation refers to "a process of taking part and also to the relations with others that reflect this process" (Wenger, 1998, p. 55). Participation may be observed in a variety of tasks which involves everything a student does and feels when engaged in a task (Hrastinski, 2009). In programming, participation could refer to a novice programmer working on a programming project, engaging in programming-related tasks during the lecture or during the practical/laboratory session, or taking part in discussion boards. Appleton et al. (2006) and Reschly and Christenson (2012) proposed participation as an indicator of behavioural engagement. Existing evidence of participation includes Shaw (2013) who reported that participation in online forums significantly improved learning scores in an ASP.Net programming course while Galyon et al. (2012) reported that students with low, medium and high academic self-efficacy predicted participation and performance. Therefore, the following hypotheses are proposed:

H2: Self-efficacy beliefs in learning programming will have a positive effect on participation.

H11a: Participation in learning programming will have a positive effect on course grade.

H11b: Participation in learning programming will have a positive effect on self-assessment.

Help-seeking

Help-seeking refers to "an achievement behavior involving the search for, and employment of a strategy to obtain success" (Ames & Lau, 1982, p. 414). Linnenbrink and Pintrich (2003) explained that help-seeking may be interpreted as a positive indicator of engagement when the student seeks help to learn and understand. However, if the student seeks help to avoid learning, then this indicator could be a negative form of help-seeking.

Karabenick (2003) proposed six types of help-seeking strategies. They are *instrumental (or adaptive) help-seeking, executive help-seeking, help-seeking threat, help-seeking avoidance, formal*, and *informal help-seeking*. Instrumental help-seeking refers to a help-seeking strategy where students ask for helpful hints from their peers or teachers instead of the answers (Nelson-Le Gall & Glor-Scheib, 1985). By contrast, students using an executive help-seeking strategy ask for help in order to avoid work and save time (Nelson-Le Gall & Glor-Scheib, 1985). Karabenick and Newman (2006) explained that adaptive help-seeking refers to a strategy where students ask for appropriate help when required. This strategy may be similar to an instrumental help-seeking strategy. Help-seeking threat and help-seeking avoidance are related as students who feel threatened about seeking help may avoid seeking help (Karabenick, 2003). Formal help-seeking strategy refers to students seeking help from the teacher as opposed to informal help-seeking where students seek help from informal sources such as peers (Gross & McMullen, 1983; Karabenick, 2003).

For the purpose of this study, help-seeking is interpreted as novice programmers who seek help from peers or teachers in order to learn and understand programming. The desired help-seeking strategy in this study would be the instrumental help-seeking strategy while help-seeking avoidance and help-seeking threat are strategies that are to be avoided. Ryan and Pintrich (1998) reported that a positive relationship exists between self-efficacy and instrumental help-seeking. In a more recent research, Ryan and Shin (2011) reported that self-efficacy contributed to help-seeking behaviour. Therefore, the following hypothesis is proposed:

H3: Self-efficacy beliefs in learning programming will have a positive effect on help-seeking.

Help-seeking improves performance in a task (Nelson-Le Gall & Glor-Scheib, 1985). In addition, Karabenick and Newman (2006) and Bembenutty and White (2013) reported that self-efficacy and adaptive help-seeking is positively associated with course grade, while according to Karabenick (2003) adaptive help-seekers achieved higher grades compared to avoidant help-seekers who performed poorly. Therefore, the following hypotheses are proposed:

H12a: Help-seeking in learning programming will have a positive effect on course grade.

H12b: Help-seeking in learning programming will have a positive effect on self-assessment.

Effort

Effort refers to the overall amount of effort expended in the process of studying (Zimmerman & Risemberg, 1997). This may refer to the extent in which a student commits his time and energy in studying or accomplishing a given task. Self-efficacy has a strong influence on *effort* (Bandura, 1977). An individual's belief in performing a task successfully is motivated by their effort, persistence, and behaviour (Bandura, 1977; Pintrich & Schunk, 1996; Zimmerman, 2000). It is expected that programming self-efficacy will have an effect on the novice programmer's effort. Therefore, this study proposes the following hypothesis:

H4: Self-efficacy beliefs in learning programming will have a positive effect on effort.

In the published literature on computer programming, McKinney and Denton (2004) and Ventura (2005) found that effort is significantly correlated to programming performance, and the findings from the literature on learning theory reported that effort had a strong effect on academic achievement (Diseth et al., 2010; Dupeyrat & Mariné, 2005; Liu, Cheng, Chen, & Wu, 2009; McClure et al., 2011; Weiner, 1985). Therefore, the following hypotheses are proposed:

H13a: Effort in learning programming will have a positive effect on course grade.

H13b: Effort in learning programming will have a positive effect on self-assessment.

Persistence

Persistence refers to a student's continued undertaking of an academic task despite facing obstacles or setbacks (Bye, Pushkar, & Conway, 2007). Self-efficacy has a strong influence on persistence (Bandura, 1977). Individuals who believe that they are able to perform a task, and can perform the task successfully, are motivated in terms of their effort, persistence and behaviour (Bandura, 1977; Pintrich & Schunk, 1996; Zimmerman, 2000;), and Brown et al. (2008) found that self-efficacy has an effect on persistence. Therefore, this study proposes the following hypothesis:

H5: Self-efficacy beliefs in learning programming will have a positive effect on persistence.

In the literature on learning theory, White (2004) found that students who persisted in learning achieved higher grades in their Information Technology course, while Glastra, Hake, and Schedler (2004) reported that persistence has an influence on the performance of the student. Therefore, the following hypotheses are proposed:

H14a: Persistence in learning programming will have a positive effect on course grade.

H14b: Persistence in learning programming will have a positive effect on self-assessment.

3.2.3.3 Cognitive Engagement

Cognitive engagement refers to the student's investment in learning, motivation to learn, and use of strategies for learning (Sheard, Carbone, & Hurst, 2010; Yazzie-Mintz & McCormick, 2012). This study focuses on the strategies used for learning programming as a measure of cognitive engagement. This is because evidence from the literature suggests that the type of learning strategy used is correlated with the performance of the student.

The following higher-order hypotheses for cognitive engagement are proposed:

HPSEC: Self-efficacy beliefs in learning programming will have a positive effect on cognitive engagement.

HCPG: Cognitive engagement in learning programming will have a positive effect on course grade.

HCSA: Cognitive engagement in learning programming will have a positive effect on self-assessment.

The learning strategies proposed to measure cognitive engagement are *deep learning*, *surface learning*, and *trial and error*.

Deep Learning and Surface Learning

Deep learning and surface learning were defined in Chapter 2, Section 2.1.4.2 (Learning Approach). The literature on learning theory generally reports positive correlations between self-efficacy, deep learning and academic achievement, and negative correlations between self-efficacy, surface learning and academic achievement (Fenollar, Román, & Cuestas, 2007; Phan, 2011). Additionally, Schunk and Mullen (2012) suggest that self-efficacious learners are likely to use learning strategies that they believe will lead to success while Pintrich and De Groot (1990) and Schunk and Mullen (2012) stated that self-efficacy influences cognitive engagement. A similar outcome is anticipated in novice programmers.

The following hypotheses are then proposed:

H6: Self-efficacy beliefs in learning programming will have a positive effect on deep learning approaches.

H7: Self-efficacy beliefs in learning programming will have a negative effect on surface learning approaches.

In the research on learning programming, a strong negative correlation was found between surface learning and performance, while a positive correlation was found between a deep approach to learning and performance (de Raadt et al., 2005; Diseth et al., 2010; Hughes &

Peiris, 2006; Miller et al., 1996; Simon et al., 2006; Yip, 2012). However, Hughes and Peiris (2006) noted that the correlation between deep learning and performance was weak. Similarly, in the literature on learning theory, Miller et al. (1996) and Yip (2012) found that different study strategies correlate with performance. Therefore, a similar finding is anticipated in this study and the following hypotheses are proposed:

H15a: A deep learning approach in learning programming will have a positive effect on course grade.

H15b: A deep learning approach in learning programming will have a positive effect on self-assessment.

H16a: A surface learning approach in learning programming will have a negative effect on course grade.

H16b: A surface learning approach in learning programming will have a negative effect on self-assessment.

Trial and Error

Trial and error is a strategy for learning where "a person tries out new strategies, rejects choices that are erroneous in the sense that they do not lead to higher payoffs" (Young, 2009, p. 626). The studies conducted by Ebrahimi (2012), Blikstein (2011), Dorn and Guzdial (2010), and Edwards (2004) suggested that programming students use a trial and error approach to debug the errors in their programming code, while other researchers have observed that students tend to use a trial and error strategy for learning in general (Carlson & Skaggs, 2000; Matzat & Sadowski, 2012).

However, there is no evidence in the existing literature on computer programming and learning theory which suggests that trial and error is an indicator of cognitive engagement, and if there is a relationship between self-efficacy and the use of a trial and error strategy. Nevertheless, since the findings from existing literature suggest that students use a trial and error strategy to learn, the following hypotheses are proposed:

H8: Self-efficacy beliefs in learning programming will have a positive effect on trial and error.

H17a: A trial and error strategy in learning programming will have a positive effect on course grade.

H17b: A trial and error strategy in learning programming will have a positive effect on self-assessment.

3.2.3.4 Emotional Engagement

Other terms used to refer to the emotional engagement dimension include *affective* (Reschly & Christenson, 2012), *motivational* (Linnenbrink & Pintrich, 2003) and *psychological* (Appleton et al., 2006). Emotional engagement refers to the student's feeling, attitude, and perception towards learning, and the learning environment (Sheard et al., 2010; Yazzie-Mintz & McCormick, 2012) and is "presumed to create ties to an institution and influence willingness to do the work" (Fredricks et al., 2004, p. 60).

The following higher-order hypotheses for emotional engagement are proposed:

HPSEE: Self-efficacy beliefs in learning programming will have a positive effect on emotional engagement.

HEPG: Emotional engagement in learning programming will have a positive effect on course grade.

HESA: Emotional engagement in learning programming will have a positive effect on self-assessment.

The indicators proposed for emotional engagement in learning programming are *interest* and *enjoyment*.

Interest

Interest is a basic emotion which motivates a student to stay focused, attentive to a learning task, and to be receptive to information (Bye et al., 2007; Dougherty, Abe, & Izard, 1996). Interest is defined as "an emotion that arouses attention to, curiosity about, and concern with…" a discipline of study (Akbulut & Looney, 2007, p. 68). Silvia (2003) suggests self-efficacy affects *interest* indirectly, where "self-efficacy affects uncertainty about how the activity will resolve, which in turn affects interest" (p.239), while Linnenbrink and Pintrich (2003) and Rottinghaus, Larson, and Borgen (2003) found that as the student develops expertise in a given task, his or her self-efficacy beliefs and interest increase. Additionally, Bandura (1997) suggests that a moderate level of self-efficacy is essential for sustaining interest in a given task, while Akbulut and Looney (2007), and Wiedenbeck et al. (2007) found a positive relationship between programming self-efficacy and interest. Therefore, the following hypothesis is proposed:

H9: Self-efficacy beliefs in learning programming will have a positive effect on interest.

McKinney and Denton (2004) and Wiedenbeck et al. (2007) found that interest is significantly correlated to programming performance. On the other hand, Sheard et al. (2010) suggest that a student's level of interest can vary across courses, with students reporting high interest in topics such as programming and computer networks. Further, in the literature on learning theory, Bye et

al. (2007) found that interest is a strong predictor of motivation to learn. Therefore, the following hypotheses are proposed:

H18a: Interest in learning programming will have a positive effect on course grade.

H18b: Interest in learning programming will have a positive effect on self-assessment.

Enjoyment

White (1964) described enjoyment as "attention to its object… an experience…and to have one's desires satisfied while doing" a task (p. 325-326). Davis (1982), on the other hand, explains that enjoyment "causes the subject to experience pleasure by causing occurrent beliefs which satisfy desires concerning the experience itself" (p. 240).

In the literature on learning theory, positive correlations were observed between self-efficacy and enjoyment (Mills et al., 2007), and that high self-efficacy results in pleasant emotions such as enjoyment (Pekrun et al., 2004; Putwain, Sander, & Larkin, 2013).

Thus, the following hypothesis is proposed:

H10: Self-efficacy beliefs in learning programming will have a positive effect on enjoyment.

In the research on learning programming, Bishop-Clark et al. (2007) found that using the Alice programming environment increased enjoyment in programming, while Liebenberg, Mentz, and Breed (2012) found that pair programming increased enjoyment in programming. In another study, Chen and McGrath (2003) found that enjoyment is an indicator of engagement when designing hypermedia while Pekrun, Goetz, Titz, and Perry (2002) and Frenzel et al. (2007) found positive correlations between enjoyment and performance. Therefore, the following hypotheses are proposed:

H19a: Enjoyment in learning programming will have a positive effect on course grade.

H19b: Enjoyment in learning programming will have a positive effect on self-assessment.

3.3 Confounding variables in this study

Although this study proposes to examine the relationship between self-efficacy, student engagement, and programming performance, it may be possible that other factors may influence the programming performance of novice programmers. These factors, also known as confounding variables, are variables that may be "mixed up with the independent variable, making it impossible to determine which of the variables has produced changes in the dependent variable" (Stangor, 2011, p. 231). Existing research on novice programmers has examined a number of factors that are likely to be predictors of programming success. These factors and the extent of their influence on programming success are discussed in Chapter 2, Section 2.1.5.3.

Based on the findings of existing research in Chapter 2, Section 2.1.5.3, this study proposes *prior programming experience* and *intelligence* as two confounding variables that are likely to influence the programming performance of novice programmers.

Prior programming experience

Prior programming experience is proposed as a confounding variable in this study since it is possible that the participants in this study may have some or considerable programming experience although they are not expected to have any programming experience when they enroll in their introductory programming course. In addition, prior programming experience is proposed as a confounding variable in this study since the findings from existing research suggest that prior programming experience may have an effect on the programming performance of the novice programmer. Prior programming experience is one attribute that predicts self-efficacy, and self-efficacy then affects the programming performance of the novice programmer (Ramalingam et al., 2004; Wiedenbeck, 2005), while the effect of prior programming experience on the programming success of the novice programmer varied from one study to another (Chapter 2, Section 2.1.5.3).

Intelligence

The intelligence of the novice programmer is the second confounding variable that is proposed in this study and is measured by the novice programmer's school results or entry qualifications into the course. The literature review in Chapter 2, Section 2.1.5.3 showed that the effect of the novice programmer's Mathematics ability on their programming performance varied from one study to another. This inconclusive finding led to a discussion with two experts who design and teach introductory programming courses. They were asked what they felt would be appropriate to measure the novice programmer's ability. During the discussions, the experts felt that the novice programmer's intelligence ie. their capacity to learn and understand may be a possible measure of their ability. As a result of the discussion, the existing literature on learning theory was examined and it was found that intelligence is a strong predictor of academic performance (von Stumm, Hell, & Chamorro-Premuzic, 2011).

Next, the literature on learning theory and computer programming was examined for possible measures of intelligence. As a result, the novice programmer's school results or entry qualifications into the course is proposed as a measure of intelligence in this study. Table 3.1 presents a list of possible measures of intelligence that were used in prior research in the field of programming or in other non-programming related fields (labelled as General) and proposes whether to accept, reject or consider using the measure in this study.

The following criteria were used to decide if a measure should be accepted, rejected, or considered as a measure of intelligence:

1. The measure should fit the interpretation of intelligence in this study ie. the novice programmer's capacity to learn and understand in general; and
2. Evidence in existing literature that the measure might have an influence on the programming performance of the novice programmer; and
3. The time and effort required by the participants to answer questions that relate to their intelligence or to provide the data required to measure intelligence.

A number of variables have been used to measure the ability of the students in Programming and Computer Science related courses. The measures include programming aptitude, comparison of grades with other courses, entry qualifications, and mathematical ability. The programming aptitude tests appeared to be a promising test to measure intelligence. However, these programming aptitude tests are licensed and the duration of these tests make it impractical for this study. This is because the participants in this study will be required to participate in two surveys during their introductory programming course (Chapter 4). Therefore, including a programming aptitude test may discourage the novice programmers from participating in the research due to the amount of time and effort that will be required to take two surveys and a programming aptitude test.

The mathematical ability of the novice programmer appeared to be a possible measure of intelligence in this study. After further consideration, it was felt that the mathematical ability of the novice programmer may not be an appropriate measure of intelligence since it has been extensively tested in prior studies and the findings were not consistent from one study to another. In addition, using mathematical ability alone as a measure of intelligence may not be a measure that would accurately reflect intelligence.

On the other hand, two measures which have received little attention in the research on computer programming are the school results (or entry qualifications) of the novice programmers and a comparison of their performance in programming with the other courses in their field of study. These two measures appear to be suitable measures of intelligence since the schools results and the comparison of performance in programming with the other courses are objective measures of the novice programmer's capacity to learn and understand. Although the finding by Alexander et al. (2003) suggests that pre-University results did not predict the novice programmer's success in programming, their findings also suggest that a student with a prior record of success is likely to succeed in the present course that they are studying. Based on this second encouraging finding, this study proposes to use the novice programmer's school results as a measure of intelligence. This study does not propose to compare the programming grades with the grades of other academic courses that the novice programmers has taken since the novice programmers may be

enrolled in a number of courses, and the courses may differ between each novice programmer. Thus, the consolidation and analysis of the results from other courses may not be feasible.

Intelligence may also be measured by general intelligence or personality traits tests. In research that is not related to the Programming and Computer Science fields, von Stumm et al. (2011) found that intelligence is a strong predictor of academic performance while Poropat (2009) argued that the academic performance of the student was independent of their intelligence and that personality traits such as those listed in the results column for Poropat's (2009) paper in Table 3.1 may have a stronger correlation with academic performance. Further, Poropat (2009) argued that if the student does not engage in the course despite having a high level of intelligence, the effect of the student's intelligence on their performance may be limited. Within the context of this study, this might imply that although the novice programmer's general intelligence may be a predictor of programming performance, engagement could instead be a stronger predictor of programming performance. Thus, Poropat's (2009) argument suggests that the general measures of intelligence are a less likely indicator to measure intelligence within the context of this study. In addition, the time, cost to purchase, and administer the general intelligence or personality trait test may not be feasible for this study as the administration of the general intelligence or personality trait test may be time-consuming and may require a purchase agreement in order to administer the test to the participants in this study. Further, these tests may discourage the novice programmers from participating in the research due to the amount of time and effort that will be required to take two surveys and an intelligence or personality traits test.

Table 3.1: Possible measures for intelligence

Field	Author(s)	Measure	Results	Decision
Programming	Caspersen, Larsen, & Bennedsen (2007)	Programming aptitude	Findings were not conclusive. Programming aptitude could not be determined from their test instrument.	Reject
Programming	Glorfeld & Fowler (1982); Wolfe (1977)	Programming aptitude	Results showed a 75% predictive power of the student's ability to think logically and concluded that model would be useful for counseling students.	Reject
Programming	Simon et al. (2006)	Comparison with other courses	Merely anecdotal evidence of performance in programming does not correlate well with performance in other academic courses.	Consider
Computer Science & Programming	Alexander et al. (2003)	Entry qualifications	Pre-university results do not predict success in Computer Science. But good pre-university results likely to predict better grades in mathematical related courses.	Accept
Programming	Ventura (2005)	Mathematical ability	SAT math scores and Critical Thinking had **little** predictive value.	Consider
Programming	Bergin & Reilly (2006); Wilson & Shrock (2001)	Mathematical ability	In separate studies, 13 researchers examined Mathematics scores and all 13 reported **strong** correlations between Mathematics and programming performance.	Consider
General	von Stumm et al. (2011)	Intelligence & Personality Trait	Intelligence is the single most powerful predictor of academic performance.	Consider
General	Flynn (1984)	Intelligence & Personality Trait	Proposed the Flynn effect – Improved environmental conditions and our cognitive ability to adapt to the environment may be a factor that explains the substantial increases in the IQ of individuals over the years.	Consider
General	Poropat (2009)	Intelligence & Personality Trait	Academic performance correlated significantly with personality traits such as: Agreeableness, Conscientiousness, and Openness. Correlations between Conscientiousness and academic performance were largely independent of intelligence.	Consider

3.4 Formative and Reflective Constructs

The decision to measure a construct reflectively or formatively should depend on how the researcher conceptualizes the construct and the objective of the study (Hair, Hult, Ringle, & Sarstedt, 2014; Mackenzie, Podsakoff, & Podsakoff, 2011) as this would have implications for the setup of the measurement model, the development of the measurement scale, and the types of analysis to perform during the data analysis stage.

According to Hair, Black, Babin and Anderson (2010):

> A reflective measurement theory is based on the idea that latent constructs cause the measured variables and that error results in an inability to fully explain the measured variable. Thus, the arrows are drawn from latent constructs to measured variable. A formative measurement theory is modelled based on the assumption that the measured variables cause the construct. The error in formative measurement model, therefore, is an inability of the measured variables to fully explain the construct. (p. 679)

Thus, the following discusses whether the three main constructs in this study should be measured reflectively or formatively. For illustrations of the first-order and second-order construct, refer to Chapter 7, Section 7.9.

Programming self-efficacy

In Chapter 6, this study proposed to measure programming self-efficacy using the scale that was developed by Ramalingam and Wiedenbeck (1998). The indicators of programming self-efficacy will be measured reflectively based on existing studies that have used the programming self-efficacy scale (Askar & Davenport, 2009; Wiedenbeck et al., 2007). The same scale will be used to measure the pre- and post-programming self-efficacy constructs.

The next consideration was in the measurement of the lower-order (or first-order) and higher-order (or second-order) constructs. In their study, Ramalingam and Wiedenbeck (1998) found that their programming self-efficacy scale produced four factors when they performed an Exploratory Factor Analysis (EFA). The four factors are simple programming tasks, independence and persistence, complex programming tasks, and self-regulation. Their finding suggested that the programming self-efficacy scale had two levels of abstraction. The first level of abstraction was the four factors of pre- and post-programming self-efficacy and the second level of abstraction was the *pre-* and *post-programming self-efficacy* constructs.

Jarvis, MacKenzie, and Podsakoff (2003) proposed four types of multi-dimensional models. The programming self-efficacy construct is proposed to be in *Type I – Reflective First-Order, Reflective Second-Order* category, whereby the indicators of the first-order constructs are the four factors of the pre- and post-programming self-efficacy constructs, and the second-order constructs are the pre- and post-programming self-efficacy constructs.

The first-order and second-order constructs are proposed to be measured reflectively since existing research on programming self-efficacy had measured the construct reflectively. In addition, the constructs are proposed to be measured reflectively since the constructs explain the indicators (Fornell & Bookstein, 1982).

Engagement

Engagement is a multi-dimensional construct which is made up of several indicators. Like the programming self-efficacy construct, the engagement construct too has two levels of abstraction. The first level of abstraction is the indicators of the multi-dimensional engagement construct. The proposed indicators of the multi-dimensional engagement construct are *participation*, *effort*, *help-seeking*, *persistence*, *deep learning*, *surface learning*, *trial and error*, *interest*, and *enjoyment*. The second level of abstraction is the *behavioural engagement*, *cognitive engagement*, and *emotional engagement* constructs.

Based on Jarvis et al.'s (2003) illustration of four types of multi-dimensional models, the engagement construct is proposed to be in *Type II – Reflective First-Order, Formative Second-Order* category. The indicators of the first-order constructs are the indicators of the second-order engagement constructs, *behavioural engagement*, *cognitive engagement*, and *emotional engagement*. The first-order constructs are proposed to be measured reflectively since the literature on student engagement explained that the indicators of engagement are reflective of each engagement dimension (Appleton et al., 2008).

The second-order constructs are proposed to be measured formatively since the higher-order constructs explain a combination of the indicators (Fornell & Bookstein, 1982). In addition, the direction of the causality is from the indicators to the construct, and changes in the indicators are expected to cause a change in the construct (Jarvis et al., 2003). These characteristics are typically demonstrated by a formatively measured construct.

Programming Performance

The two constructs to measure programming performance – *programming grade* and *self-assessment* will be measured reflectively as the constructs are intended to explain the indicators (Fornell & Bookstein, 1982).

3.5 Research model with Hypotheses

Based upon the discussion of the literature review in Chapter 2 and the discussion of the research model in this chapter, Figure 3.2 presents the research model, the relationships, and the hypotheses proposed in this study. The hypotheses on the left side of the research model (H1 – H10) seek to address Research Question 1 (RQ1), and the hypotheses on the right side of the research model (H11 – H19) seek to address Research Question 2 (RQ2). The proposed confounding variables are also presented in the research model, in the box with dotted lines on the top right of the research model.

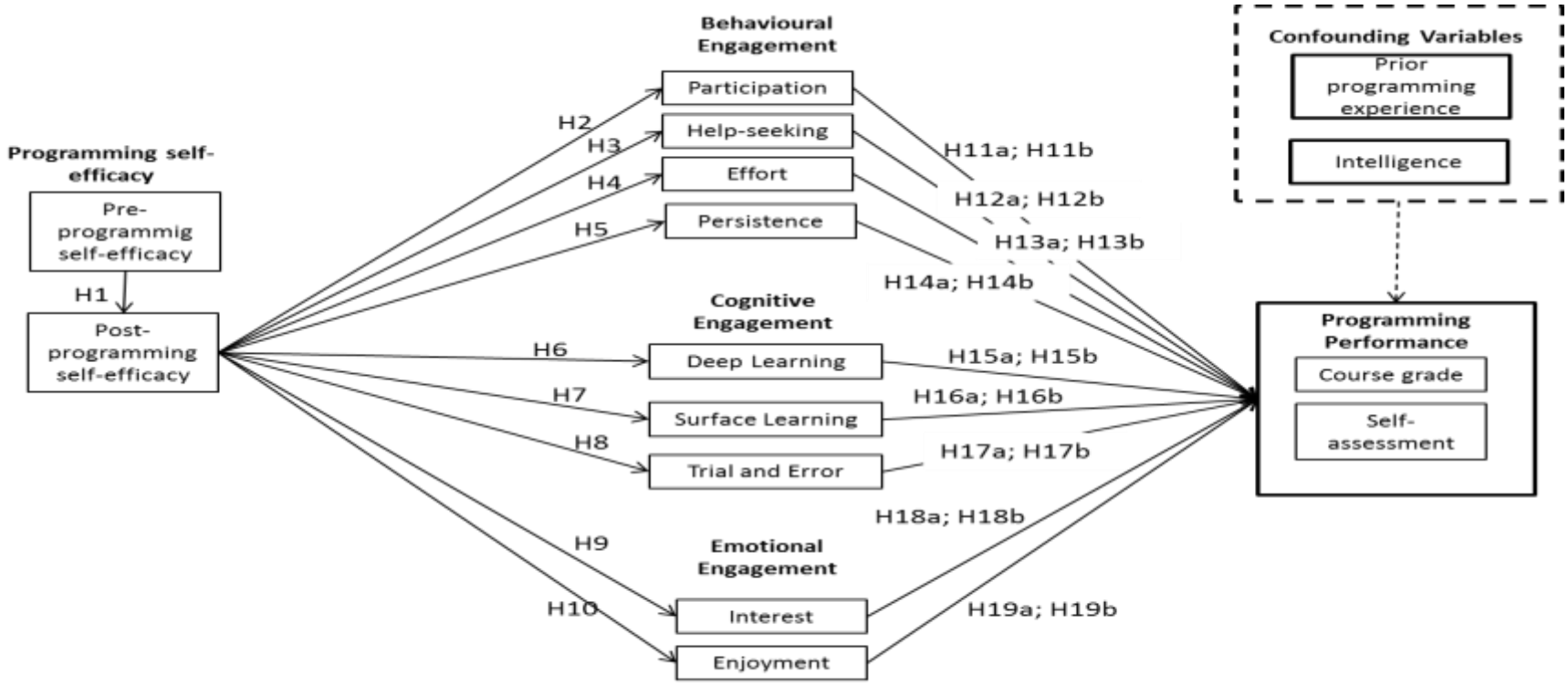


Figure 3.2: Research model with hypotheses, confounding variables, and indicators of behavioural, cognitive, and emotional engagement

3.6 Chapter Summary

The purpose of this chapter was to develop the conceptual research model that was used to address the research questions in this study. The initial research model presented three key constructs – programming self-efficacy, engagement, and programming performance (Figure 3.1).

The pre- and post-programming self-efficacy constructs were proposed as indicators of the programming self-efficacy construct. Since these constructs had two levels of abstraction, a reflective-reflective measurement model was proposed based on evidence from existing research. Hypotheses were developed for this construct and were illustrated in the research model in Figure 3.2.

The multi-dimensional engagement construct was proposed to be made up of behavioural engagement, cognitive engagement, and emotional engagement. The indicators of behavioural engagement are proposed to be participation, help-seeking, persistence, and effort, while the indicators of cognitive engagement are proposed to be deep learning, surface learning, and trial and error. The indicators of emotional engagement are proposed to be interest, and enjoyment. Since the engagement construct had two levels of abstraction, a reflective-formative measurement model was proposed based on evidence from existing research, and based on the conceptualization of the engagement construct in this study. Hypotheses were developed for this construct and were illustrated in the research model in Figure 3.2.

Two indicators were proposed for the programming performance construct. They are programming grade and self-assessment. Both constructs were proposed as reflective constructs based on the conceptualization of the programming performance construct in this study. Hypotheses were developed for this construct and were illustrated in the research model in Figure 3.2.

Finally, prior programming experience and intelligence were proposed as confounding variables in this study and were illustrated in the research model in Figure 3.2.

Chapter 4: Methodology

Crotty (1998) argued that the choice of methodology and methods used in a research should be guided by the theoretical perspective or theoretical paradigm of the study. The discussion in this chapter reflects on the theoretical paradigm that is applied in this research and justifies the methodology and methods used in this research by evaluating the epistemological and ontological perspectives of research. The methods that may be used to measure engagement are then compared and justifications are made for the use of self-reports and interviews to measure student engagement. Finally, the research method, research approach and ethical considerations are discussed.

4.1 Research Paradigm

In any research, the researcher is guided by a set of beliefs which is referred to as *epistemology* and *ontology* (Denzin & Lincoln, 2011; Guba & Lincoln, 1994). Epistemology is concerned "with the nature of knowledge and the proper methods of inquiry" (Iivari, Hirschheim, & Klein, 1998, p. 174), and provides a view by which people understand knowledge or "how we know what we know" (Crotty, 1998, p. 8). Ontology is "concerned with the structure and properties of what is assumed to exist, i.e. the basic building blocks that makes up the phenomena or objects to be investigated" (Iivari et al., 1998).

Positivism, *interpretivism* and *critical theory/inquiry* are three common views that underpin epistemology in social science research (Denzin & Lincoln, 2005; Crotty, 1998). A positivist assumes the existence of an a priori relationship with the phenomena being observed and the relationship may be investigated using structured instrumentation (Orlikowski & Baroudi, 1991). In particular, positivist research in Information Systems (IS) assumes that the issue being examined may contain propositions, variables which can be measured and quantified, hypotheses that can be tested, and uses a sample of the population to draw inferences from the study (Orlikowski & Baroudi, 1991). Similarly, Gregor (2006) argued that a positivist research uses theory as a means to explain, predict, and test a phenomenon. Positivist research is more common in IS research compared to interpretivist and critical theory research paradigms (Orlikowski & Baroudi, 1991). The characteristics of positivist research are closely associated with the beliefs of the researcher in this study and it is therefore adopted as a theoretical paradigm to develop a research model, and to explain, predict and test the relationship between self-efficacy, engagement, and the programming performance of the novice programmer.

By contrast, an interpretivist research approach "assumes that people create and associate their own subjective and intersubjective meanings as they interact with the world around them" (Orlikowski & Baroudi, 1991, p. 5), and tends to use interpretive case studies or ethnography (Denzin & Lincoln, 2011, p. 13), while a critical theory research assumes that

the researcher and the phenomenon being researched are linked to the values of the researcher, resulting in the researcher influencing the research inquiry (Guba & Lincoln, 1994) and is narrated in the form of historical, economic, and sociocultural analyses (Denzin & Lincoln, 2011, p. 13).

Ontology in IS research relates to the "information and data, information systems, human beings in their different roles of IS development and IS use, technology, human organizations and society at large" (Iivari et al., 1998, p. 172), which provides strong support for this study which is concerned with the study of human beings (the novice programmer's self-efficacy and engagement), information systems, and technology (programming). Further, the ontological perspective of a positivist research assumes that the "physical and social world exists independently of humans" (Orlikowski & Baroudi, 1991, p. 9). Typically, a positivist researcher studies a problem or phenomenon by constructing a model and measuring the constructs in the model using a set of instruments suitable for examining the phenomenon. The positivist ontological perspective is closely associated with the researcher in this study.

4.2 Comparison of Methods to Measure Student Engagement

Table 4.1 lists the methods that may be used to measure student engagement and their informing references. The "+" symbol lists the advantage of the method, and the "-" symbol lists the disadvantage of the method. Based on Table 4.1, student self-reporting is clearly the most popular method for measuring student engagement, particularly for measuring cognitive and emotional engagement. Interviews, too, appear to be feasible for this study particularly when the indicators of engagement in introductory programming courses need to be refined and validated.

The use observation as a method to measure engagement appears to have more disadvantages than advantages, and teacher rating does not appear to be feasible in this study since this method may not be appropriate to gather data on the emotional engagement of the novice programmers. In addition, experience sampling is not a suitable method in this study since the large time investment that is expected from the participants may not be practical for novice programmers as they are expected to have a heavy workload at the tertiary level.

Therefore, having weighed the advantages and disadvantages of each method, and since this study has adopted a behavioural research paradigm to examine the relationships between self-efficacy, engagement and the programming performance of the novice programmer, this study proposes to use a self-reporting method (survey questionnaires) to measure the novice programmer's level of engagement, and group interviews (focus groups) to identify and confirm the proposed indicators of engagement in introductory programming courses.

Table 4.1: Comparison of Methods to Measure Student Engagement

Measurement Method		Description	Informing References
Self-report	+	Most common method	Appleton et al. (2006); Fredricks & McColskey (2012)
	+	Useful for student's subjective perception	
	+	Useful for examining emotional and cognitive engagement which are not directly observable	
	+	Practical and easy to administer in a classroom	
	+	Administered to large samples	
	–	Responses may not be honest	
	–	Could contain items that are broadly worded and not reflective of the indicator of engagement	
Experience sampling	+	Allows collection of detailed data at the precise moment	Fredricks & McColskey (2012); Hektner, Schmidt, & Csikszentmihalyi (2007)
	+	Reduces recall failure	
	+	Reduces responses which may be socially acceptable	
	+	Able to collect information across time and situations	
	–	Large time investment by students	
	–	Success highly dependent on participants' ability and willingness	
	–	Suitable for measuring small number of engagement items	
Teacher Ratings	+	Useful for triangulating teacher and student responses to behavioural engagement	Fredricks & McColskey (2012); Skinner et al. (2008)
	–	Useful for studies involving younger children	
	–	Might not be useful for measuring emotional engagement	
Interview	+	Provide insights into variations in the levels of engagement	Fredricks & McColskey (2012); McCaslin & Good (1996)
	+	Detailed and descriptive account of experiences, engagement, and contextual factors	
	–	Biased view due to interviewer	
	–	Reliability and validity of findings	
	–	Concerns about social desirability	
Observations	+	Verify data collected from surveys and interviews	Fredricks et al. (2004); Fredricks & McColskey (2012)
	–	Time consuming	
	–	Data may need to be collected in various settings (group work, seatwork)	
	–	Reliability issues if observer is not well trained	
	–	Limited information on the quality, effort, participation, or thinking	
	–	Involves a small number of students	

4.3 Research Method

Quantitative, *qualitative* and *mixed methods* are three research approaches for conducting research in the field of social sciences (Bryman, 2008; Denzin & Lincoln, 2005). This study uses a mixed methods approach - where both quantitative and qualitative approaches are used to address the objectives of this study. The advantage of using a mixed methods approach includes the ability to provide complementary divergent views of the phenomenon being studied, the strength of the inferences are increased using two methods, and are able to answer confirmatory and exploratory research questions at the same time (Teddlie & Tashakkori, 2009). Further, Venkatesh, Brown, and Bala (2013) identified seven purposes of mixed methods research. This study fits into the *developmental* purpose of mixed methods research.

A developmental mixed methods research may require a qualitative study that will assist in the development of constructs and hypotheses that will then be tested in a quantitative study (Venkatesh et al., 2013, p. 26). Based on evidence from existing literature on computer programming and learning theory, this study proposed a research model and a set of hypotheses to test the relationships between programming self-efficacy and engagement, and between engagement and the programming performance of the novice programmer in an introductory programming course (Chapter 3). The indicators of engagement were then identified based on evidence from existing literature on computer programming and learning theory. A qualitative study in the form of focus groups was then used to refine and validate the indicators of engagement in introductory programming courses (Chapter 5), followed by a quantitative study in the form of survey questionnaires was used to test the hypotheses and validate the research model (Chapter 6 & Chapter 7).

4.4 Research Approach

The approach used to develop and validate the research model adheres to the guidelines proposed by Hong, Chan, Thong, Chasalow, and Dhillon (2014). They proposed six guidelines for developing context specific theory in IS research. In the first guideline, they suggested that the research could be built on general theory while the second guideline states that the model could then be refined to include constructs that are only specific to the research. In the third and fourth guideline, they suggested that the context-specific factors could be further refined using, for example, qualitative analysis and can then be included in the research model. In the fifth guideline, the interaction between the context-specific factors should be examined, and finally, the sixth guideline may be applicable if there is a need to examine alternative context-specific models such as mediation. Based on guidelines 1 and 2, this study developed a research model in Chapter 3. Next, using guidelines 3, 4 and 5, this study proposed a three-phased approach to test and validate the research model which is discussed in the following sub-sections.

4.4.1 Phase 1: Survey Questionnaire – *Pre-programming self-efficacy*

Hypothesis 1 in the research model in Figure 3.2 (Chapter 3) proposed that the post-programming self-efficacy scores will be higher than the pre-programming self-efficacy scores when learning programming. To examine this relationship, a survey research using self-reporting questionnaires was administered to novice programmers at the beginning of their introductory programming course (typically in the third week of the course). A survey research was used as it is a common method for gathering quantitative data in IS research (Pinsonneault & Kraemer, 1993). Additionally, a survey was appropriate in this study since the data had to be collected from a sample of novice programmers so that the findings may be generalized to the larger population of novice programmers (Pinsonneault & Kraemer, 1993; Punch, 2005).

The survey questionnaire was also used to collect data for prior programming experience, which is one of the two confounding variables in this study. The data collection for intelligence, which is the second confounding variable in this study, was measured by the novice programmer's school results. The school results of the participants were obtained from their institution's student records after obtaining approval from the participants.

4.4.2 Phase 2: Focus Groups – *Construct Validation*

The focus groups were held halfway through the first introductory programming course (in week 7 of a 14-week course). The purpose of the focus groups was to refine and validate the indicators of the engagement construct that was proposed in the research model (Chapter 3) and to identify new indicators of engagement in introductory programming courses. The participants in the focus groups were novice programmers who were enrolled in an introductory programming course.

4.4.3 Phase 3: Survey Questionnaire – *Validate the research model*

A cross-sectional survey using self-reporting questionnaires was administered to the novice programmers at the end of the introductory programming course (final week of the course). The purpose of the Phase 3 survey was to validate the research model by operationalizing the research model on a large population of novice programmers. The participants that had participated in Phase 1 of the research were contacted to participate in Phase 3 of the survey.

Like Phase 1, Hypothesis 1 in the research model proposed that the post-programming self-efficacy scores will be higher than the pre-programming self-efficacy scores when learning programming. To examine this relationship, the programming self-efficacy scale that was used in Phase 1 was repeated in this phase. In addition, the Phase 3 survey questionnaire collected data on the refined engagement constructs and the novice programmer's self-assessment of their programming performance.

The programming grade data, which is the second measure for programming performance, was obtained from the participant's institution's student records. The approval to access the participant's programming grade was obtained in Phase 1 of the data collection. The Phase 1 and Phase 3 survey questionnaire was repeated on several cohorts of introductory programming courses in Malaysia and in New Zealand.

4.5 Ethics Approval

Since this research involved human participants and required access to their school results (measure for intelligence confounding variable) and programming grade (measure for programming performance), approval was obtained from the School of Information Management Human Ethics Committee at Victoria University of Wellington before contact was made with the participants. Appendix E contains the consent forms for the survey questionnaires (Phase 1 & Phase 3) and the focus groups (Phase 2).

An ethics application was submitted in each phase of this study. In Phase 1, ethics approval was obtained to administer the survey questionnaire, and to access the participant's school results and programming grade. The participants were assured of the confidentiality of the data and signed a consent form which allowed access to their school results and their programming grade. The novice programmers who were approached to participate in the research were also assured that if they chose not to participate in this study or decide to withdraw from the study at a later date, their non-participation will have no bearing on their course grade. The participants were informed that they could, at any time withdraw from the study up until the first week after the end of their introductory programming course. The consent forms and the approval to proceed with the research from the Human Ethics Committee (HEC) were attached to an accompanying letter that was addressed to the Manager who was responsible for the student records at each participating HEI for approval to access the participant's school results and programming grade.

On several occasions, ethics approval was also obtained from the participating HEIs. The HEC of several HEIs in New Zealand and in Malaysia had policies that required all external research to be approved. The HEC application forms were submitted to the participating HEI for approval.

In Phase 2, the participants in the focus groups signed consent forms that allowed the discussion to be audio recorded, and to keep the discussions in the focus group confidential. The participants were assured of the confidentiality of the data and were advised that they will not be able to withdraw from the focus groups as it would not be possible to extract their comments from the focus group recording.

Human ethics approval was also obtained for administering the survey questionnaire in Phase 3. The Phase 3 survey questionnaire was developed after analysing the data in Phase 2 and the participants had signed the consent form in Phase 1. Participants were again assured of the confidentiality of the data and that they could withdraw from the study at any time up until the first week after the end of their introductory programming course. The survey questionnaires, the participant's school results, programming grade, the audio recordings, and transcripts from the focus groups were securely stored to maintain the privacy of the participants.

4.6 Chapter Summary

This chapter examined the epistemological and ontological perspectives and adopted a positivist perspective in this study. The various methods to measure engagement were then examined. Self-reports and group interviews were found to be the most appropriate method to collect data in this study.

Finally, a three-phased mixed methods approach was used to collect data. The first phase involved a survey questionnaire to collect data on the pre-programming self-efficacy beliefs of the novice programmers. The second phase involved focus groups in the first introductory programming course to validate and refine the indicators of engagement, and the third phase involved a survey questionnaire to operationalise the research model. The first and third phases of data collection were repeated on several cohorts of introductory programming courses in New Zealand and in Malaysia. The participants of the research were novice programmers who were enrolled in an introductory programming course throughout the duration of the data collection. Human ethics approval was obtained prior to collecting the data.

Chapter 5: Focus Groups

The data collection phase commenced once the research approach was established, and human ethics approval was obtained. Although the data collection started with Phase 1 (survey), the findings of the focus groups (Phase 2) are discussed first in this thesis since the focus groups were conducted only during the first introductory programming course. The design of the survey in Phase 1 and Phase 3 are jointly discussed in Chapter 6 since the analysis of the data for these two phases are linked.

The purpose of the focus groups was to validate and refine the proposed indicators of engagement in introductory programming courses (Chapter 3, Section 3.2.3), and to identify new indicators of engagement in introductory programming courses. The focus group questions were developed and the focus group participants were recruited during the planning phase. After conducting the focus groups, the discussions were transcribed and analysed for evidence of the indicators of engagement in introductory programming courses. The findings of the focus groups led to the validation and refinement of the indicators of engagement in the research model, and the research hypotheses. In addition, the differences between introductory programming courses and other courses are discussed.

5.1 Planning for the Focus Groups

Planning for the focus groups involved the development of the questions for discussion and recruitment of the participants.

5.1.1 Development of Focus Group Questions

The questions for the focus groups were developed based on the:

1. *Definition of Student Engagement.* The definitions of each higher-order engagement construct led to the development of the questions for the focus groups (Chapter 3, Section 3.2.3).
2. *Objective of the focus groups.* As previously mentioned (Chapter 4, Section 4.4.2), the objective of the focus group is to validate and refine the proposed indicators of the student engagement construct, and to identify new indicators of student engagement in introductory programming courses. The focus group participants are asked questions that would prompt them to discuss their engagement behaviour when they are learning programming in their introductory programming course. In doing so, the analysis of the discussions may lead to the identification of specific indicators of engagement in introductory programming courses.

The proposed questions were reviewed by three academics from Victoria University of Wellington who had more than 15 years of teaching experience at the tertiary level and had experience teaching introductory programming courses.

Morgan (1997) suggested that "for more structured groups, it is useful to organize the discussion topics into a guide that the moderator follows in more or less the same order from group to group" (p. 47). As the objective of the focus groups was to validate and refine the proposed indicators of the student engagement construct, a structured discussion with a set of guided questions would ensure that the participants stayed focused during the focus group discussion. Table 5.1 lists the focus group questions for each engagement dimension.

Table 5.1: Focus group questions

No	Question	Engagement Dimension
1.	What do you do when you are stuck in your workshops / practicals or assignment?	Behavioural Engagement
2.	What activities are you involved in during lectures and workshops/practical sessions? Are these useful for learning programming?	
3.	How do you prepare for your lecture and workshops/practical sessions?	
4.	Is your method of preparation in this course different from how you prepare for your other courses? Please explain.	
5.	How do you approach/tackle your programming assignment or workshops/practical exercises?	Cognitive Engagement
6.	What strategies or techniques work for you when you are attempting to code solutions for your programming assignment or workshop/practical exercises?	
7.	What are your feelings when you are working on your workshop/practical exercises or programming assignment?	Emotional Engagement
8.	Are these feelings different from your other courses?	
9.	How do you learn programming?	General Engagement
10.	Is learning programming different from how you learn for other courses? Please explain.	
11.	What do you think are important attitudes to have in order to be successful in this course?	

5.1.1.1 Behavioural Engagement

The following definition was referred to when developing the focus group questions for behavioural engagement:

Behavioural engagement refers to the student's involvement in observable academic, social and extracurricular activities (Fredricks et al., 2004; Yazzie-Mintz & McCormick, 2012).

In the definition above, behavioural engagement extends beyond the classroom, to non-academic school-based activities. This study focuses on the academic-related activities that the novice programmers undertake when learning programming.

The questions to identify the indicators of the behavioural engagement construct (Table 5.1) were developed to understand how the novice programmers interacted and participated in their lectures and workshop/practical sessions, and the activities that they were involved in when working on their programming assessments.

5.1.1.2 Cognitive Engagement

The following definition was referred to when developing the focus group questions for cognitive engagement:

Cognitive engagement refers to the student's investment in learning, motivation to learn, and use of strategies for learning (Sheard at al., 2010; Yazzie-Mintz & McCormick, 2012).

The questions to identify the indicators of the cognitive engagement construct (Table 5.1) were developed to understand the strategies that the novice programmers used when they attempted their programming assessments, and to understand which strategies worked for the novice programmers.

5.1.1.3 Emotional Engagement

The following definition was referred to when developing the focus group questions for emotional engagement:

Emotional engagement refers to the student's feeling, attitude and perception towards learning, and the learning environment (Sheard et al., 2010; Yazzie-Mintz & McCormick, 2012) and are "presumed to create ties to an institution and influence willingness to do the work" (Fredricks et al., 2004, p. 60).

The questions to identify the indicators of the emotional engagement construct (Table 5.1) were developed to understand the feelings of the novice programmers when working on their programming assessments.

5.1.1.4 Other Engagement Questions

In addition to the questions that were based on each dimension of student engagement, general questions on learning programming were asked. Based on Table 5.1, Question 9 was asked as a general question to start the focus group discussion. Question 11 was asked as the final question to end the focus group and to have an understanding of the attitudes that the participants thought were important in an introductory programming course. Question 10 was asked to compare if the novice programmers were learning programming the same way as they were learning in other courses.

5.1.2 Recruitment of Participants

The focus group participants were novice programmers who, at the time of the data collection were enrolled in an introductory programming course in a Higher Educational Institution (HEI) in New Zealand or in Malaysia. Victoria University of Wellington (VUW)

in New Zealand, and Asia Pacific University of Technology and Innovation (APU) in Malaysia were selected for the recruitment of participants as both Universities had local and international students enrolled in their Undergraduate programmes, thereby offering an international perspective of the novice programmers' engagement in introductory programming courses. The focus groups were held mid-way through the introductory programming course. The invitations to participate were sent one week before conducting the focus group. Appendix A contains the approval letter from the School of Information Management Human Ethics Committee (HEC), Victoria University of Wellington (VUW) to proceed with the data collection. Fliers containing invitations to participate in the focus group were distributed to the novice programmers during their lectures and posted electronically on their Learning Management System (LMS).

5.2 Conducting the Focus Groups

The focus groups were conducted in English in a discussion room at the participants' HEI. The participants were comfortably seated around the table and the moderator was seated at the head of the table. At the start of the focus group, a participant information sheet and a participant consent form was given to each participant. The participants were advised to sign the consent form that required them to keep the proceedings of the focus group confidential, and that they will not be able to withdraw from the focus group at a later date as it will not be possible to extract their comments from the audio recording. Participants were then briefed that the purpose of the focus group was to understand how they were engaging in their introductory programming course.

The suggested size of a focus group varies between researchers. Researchers have suggested a focus group size that is as large as 12 (Stewart, Shamdasani, & Rook, 2007), to as small as 4 Greenbaum (1993). Table 5.2 gives a breakdown of the focus group participants. The size of each focus group was between 3 – 5 participants. The participants in the focus groups varied in their level of study (Diploma and Degree), type of major, the programming language used, and the assessment methods that was used in their introductory programming course.

The smaller number of participants in each focus group was due to scheduling constraints since the participants were occupied with the workload in their courses and had commitments outside of the University. Although the sizes of the focus groups were smaller, all the participants were highly participative throughout the focus group discussion. The participants gave detailed insights into their engagement behaviour during their introductory programming course. Each focus group took between 75 and 90 minutes and was audio-recorded. The discussions seldom deviated from the main topic of discussion, and the participants were highly interactive with little encouragement needed to offer their views. The transcripts from each focus group averaged to about 21 pages resulting in a rich set of

narratives that provided strong evidence of the participant's engagement behaviour in their introductory programming course.

Table 5.2: Breakdown of focus group participants

Focus Group #	Location	Date	School	No. of participants
Focus Group 1	VUW, NZ	September 2013	SIM	2
			SECS	3
Focus Group 2	VUW, NZ	September 2013	SIM	4
			SECS	1
Focus Group 3	VUW, NZ	September 2013	SIM	3
Focus Group 4	APU, Malaysia	January 2014	SCT	3

Legend:
SECS- School of Engineering and Computer Science
SIM - School of Information Management
SCT - School of Computing and Technology

Due to the timing of the data collection in Malaysia which coincided with the end of the year term break, and religious holidays at the start of the following year, only one focus group was conducted in Malaysia. However, like the focus groups in New Zealand, the participants from the focus groups in Malaysia gave detailed insights into their engagement in their introductory programming course. In addition, the activities that the participants were engaged in were similar to the activities of the participants in the New Zealand focus groups. Thus, it was not necessary to conduct a second focus group in Malaysia.

How many focus groups would be sufficient for this study was the next consideration when conducting the focus groups. Morgan (1997) suggested that 3 – 5 groups per project may be sufficient as more groups may not yield further insights into the research problem. Other researchers suggest that the focus groups should continue until the discussion reaches a "saturation" point (there are no new issues arising from subsequent focus groups) (Glaser & Strauss, 1967; Strauss & Crobin, 1990), while Krueger and Casey (2009) proposed that the researcher could evaluate if there is a need to hold more focus groups after the third focus group. During the analysis stage, several engagement themes emerged from focus groups 1 and 2. In focus groups 3 and 4, the same engagement themes that were identified in focus groups 1 and 2 had emerged. Therefore, 4 focus groups were sufficient to confirm the indicators of engagement in introductory programming courses.

The skills of the moderator were the final consideration in the conduct of the focus groups. Several researchers suggested that the quality of data collected from focus groups was dependent on the skills of the moderator (Stewart et al., 2007). The moderator should have prior experience in working with groups, be familiar with group processes, be comfortable, and curious about the topic and the participants (Krueger & Casey, 2009). In this study, the researcher was the ideal moderator for the focus group. The researcher has an interest in the focus group discussion and had developed her skills to be a moderator from the guidelines

suggested in focus group reference texts, and online educational videos on conducting focus groups.

5.3 Findings and Analysis of Focus Groups

The focus groups were analysed using the key concepts framework proposed by Kreuger and Casey (2009). Then, an inductive and deductive approach (Thomas, 2006) was employed for the detailed analysis of the focus groups, resulting in the identification and validation of the indicators of engagement in introductory programming courses (see Appendix B).

5.3.1 Method of Analysis

The analysis of the focus groups conforms to the *key concepts* analysis framework that was proposed by Krueger and Casey (2009). Their key concepts analysis framework identifies the key concepts or themes in the research by asking the participants to identify the core themes, important ideas, and experiences that relate to the core themes. In this study, the objective of the focus groups was to refine and validate the proposed indicators of engagement in introductory programming courses. The data from the focus groups were analysed for evidence of the proposed indicators of engagement by examining the engagement behaviour of the novice programmers.

Once the analytic framework was established, the focus group data was then analysed using an inductive and a deductive approach (Thomas, 2006). He defines the deductive approach as "…data analyses that set out to test whether data are consistent with prior assumptions, theories, or hypotheses identified or constructed by an investigator", while the inductive approach refers to "….approaches that primarily use detailed readings of raw data to derive concepts, themes, or a model through interpretation made from the raw data by an evaluator or researcher" (Thomas, 2006, p. 238).

The inductive and deductive analysis approaches were compared with three common qualitative analysis approaches. They are discourse analysis, phenomenology, and grounded theory. Discourse analysis and phenomenology was not suitable for the analysis of the focus group data in this study as discourse analysis is focused on describing the multiple meanings of the text, while phenomenology is focused on providing a narrative of the experience of the participants (Thomas, 2006, p. 241). On the other hand, the grounded theory approach identifies core themes and describes the theory (Strauss & Corbin, 1988). Since, the objective of the focus groups was to refine and validate the indicators of engagement in introductory programming courses, and not to describe the theory, the inductive and deductive analysis approach was more suitable for identifying the indicators of engagement using the focus group data.

During the analysis of the data, the deductive approach was used to examine the data for evidence of the indicators of engagement that were proposed in the research model in Figure 3.2 (Chapter 3, Section 3.5), while the inductive approach was used to code the data, and to

identify new indicators of engagement. The participants were asked focused questions that required them to describe how they were engaging in their introductory programming course. According to Thomas (2006), using an inductive approach to analyse data was suitable for research that had focused questions.

The inductive analysis approach involved reading the focus group transcripts to identify the engagement themes, which are then reduced and interpreted by identifying the indicators of engagement (Thomas, 2006). The inductive analysis approach is consistent with the three-step data analysis process of *data condensation*, *data display*, and *drawing and verifying conclusions* that were proposed by Miles, Huberman, and Saldaña (2014). The following describes the coding process using the inductive analysis approach:

Step 1: Preparation of raw data files (data cleaning)

The audio recordings of the focus groups were backed up into an external hard drive. The focus groups were then transcribed and formatted using a word document. The transcripts were then checked against the audio recordings to ensure that there were no errors in the transcription process.

Step 2: Close reading of text

The transcripts were then read in detail in order to gain an overall understanding of the engagement themes that were emerging from the transcripts. During this step, the data from the transcripts were condensed. Miles et al. (2014) suggested that one way of condensing the data was to abstract the data from interview transcripts. The decision of which data to abstract was made based on whether the abstracted data had answered the focus group questions. Discussions that deviated from achieving the objective of the focus group were not abstracted. One example of the deviation was the discussion of the teaching style of the introductory programming course instructor. The excerpts from the responses of the participants that had been abstracted from the transcripts were then grouped based on the focus group questions. This step is the first step in the coding process for qualitative data.

Step 3: Creation of categories

The next step in the analysis of the focus groups was to identify categories and themes. However, prior to the identification of the categories and themes, the decision on how to display the data had to be made. A good display of data can lead to a robust analysis of qualitative data as the researcher is able to view the data in an organized and concise manner, and draw conclusions from the data (Miles et al., 2014). The main criteria for deciding on the display of the focus group data was to enable the researcher to analyse the responses to each question, distinguish between the focus groups, and to identify the similarities in the engagement patterns of the participants in each focus group. Using these criteria, the data was displayed in a matrix using a spreadsheet.

To create categories and themes, a two-level analysis was performed. In the 1st level of analysis, similarities in the responses were consolidated and assigned codes. Miles et al. (2014) define codes as "labels that assign symbolic meaning to the descriptive or inferential information compiled during a study" (p. 71). For example, Table 5.3 presents excerpts from the responses of several participants when asked what they would do when they were stuck in their workshop/practical exercises, or in their assignments. The responses suggest that students tend to seek help from several sources when stuck in their assessments. Therefore, the responses were coded as "Seek help from < source(s)>".

Table 5.3: Example of 1st level of focus groups analysis

Response	**1st level of code**
"I couldn't finish any assignment until the third week. Until I found some help from the tutor." (YM, FG1)	Seek help from: - Tutor
"Usually, I ask MJE [peer] and Ms. A [instructor]. Most of the time they can help me. If I can't find a solution from either of them, I approach the Internet." (FA, FG4)	Seek help from: - Peer - Instructor - Internet
"I'd start at the lecture slides and obviously, attend the lectures, and then move on to previous workshops to see if there is something in there which I might have missed ...My sister. I'll go to her. Then I'll go to my dad." (RS, FG2)	Seek help from: - lecture slides - previous workshops - family members - Internet

Table 5.4: Example of identification of engagement indicators from the 1st level of code

1st level of code	**2nd level of code** **Indicators of Engagement**
Seek help from: - Tutor	Engagement indicator: Help-seeking
Seek help from: - Peer - Instructor - Internet	
Seek help from: - lecture slides - previous workshops - family members - Internet	

The 2nd level of analysis identified the indicators of engagement and other emerging themes that were related to motivation in introductory programming courses. The identification of the indicators of engagement was guided by the research model. For example, Table 5.4 presents the indicators of engagement that had emerged from the 1st level of code.

Step 4: Overlapping coding and uncoded text

During the coding of the transcripts, several responses overlapped between the indicators of engagement. For example, the responses that indicate that help-seeking is an indicator of behavioural engagement were also used as evidence that participation is an indicator of behavioural engagement. Participants who seek help when learning programming could also be interpreted as participating in the course.

Step 5: Continuing revision and refinement of category system

The first two focus groups were conducted within the same academic week (Week 6) in September 2013. The analysis of the first two focus groups revealed similar responses. Two more focus groups were held to verify the findings from the first two focus groups, and to ensure that the findings had reached a saturation point. The third focus group was conducted in Week 7, after the analysis of the first two focus groups. Four months later, a fourth focus group was held in Malaysia. No new indicators of engagement emerged from the third and fourth focus groups, and the findings from the first two focus groups could be confirmed. Therefore, the four focus groups were sufficient for the confirmation and identification of indicators of engagement in introductory programming courses.

The analysis of the focus groups was verified by two academics from Victoria University of Wellington. Both academics had more than 15 years of experience teaching introductory programming courses at the tertiary level.

5.3.2 Behavioural Engagement

Table 5.5 lists the indicators of behavioural engagement that emerged from the analysis of the focus groups:

Table 5.5: Indicators of behavioural engagement in an introductory programming course

Dimension	**Indicators**	
	Initial (from Literature Review)	**New (from focus groups)**
Behavioural Engagement	Participation	-
	Help-seeking	
	Effort	
	Persistence	

The initial research model had proposed participation, help-seeking, effort and persistence as indicators of behavioural engagement, and the analysis of the focus groups confirmed the presence of these indicators in introductory programming courses. No new indicators of behavioural engagement had emerged from the focus groups.

5.3.2.1 Participation

Table 5.6 contains the excerpts from the responses of the participants which suggested that they had participated in their introductory programming course. The focus group participants participated in their introductory programming course in several ways. The participants attended lectures, attended workshops (practicals), revised their course notes, and asked for help on their programming assessments.

Table 5.6: Participation – evidence from responses

Responses from participants	Forms of participation
"...attend lectures, which in a way handy and in a way not" (DI, FG2)	*Attendance at lectures*
"I try to go [to lectures] but I feel guilty if I don't go. But I do tend to zone out a lot." (RB, FG3)	*Attendance at lectures*
"At least I understand what each thing means then. And you go to the workshop and they explain it and how to do it. " (JR, FG3)	*Attendance at workshop (practical)*
"I pretty much only go into [the lecture] out of obligation…. I guess whatever is on the slides you can just pretty much read… if I miss a verbal snippet [lecture] I can just again Google it" (CW, FG1)	*Attendance at lectures*
"lab I use it as I need to ask my questions there.. the lab is just 1 hour so I ask, ask, ask.. there is still bugs and then I ask the tutor again to help me to check the bugs and the debug thing" (SZ, FG1)	*Attendance at workshop (practical)*
"If you go there [practicals] unprepared I'll be surprised if you can finish it [referring to workshop exercises]. Because sometimes I've gone there and I've finished ¾ of the material, and I'm still there for 2 hours trying to figure it out. " (RS, FG2)	*Attendance at lectures* *Attendance at workshop (practical)*
"I go to lectures, then I do the workshops..." (DI, FG2)	*Attendance at lectures* *Attendance at workshop (practical)*
"I attend all the lectures, every tutorial, and I will spend an hour just reading the lecture notes at night" (YL, FG1)	*Attendance at lectures* *Revising course notes*
See Table 5.7	*Seek help*

The need to participate in the introductory programming course appeared to be purposeful. The participants appeared to participate in activities that were useful for their programming assessments. For example, from the excerpts in Table 5.6, one participant, SZ, from Focus Group 1 (FG1) explained that she participated in the lab session so that she could ask questions to resolve the errors in her programming assessment. On the other hand, participant DI from Focus Group 2 (FG2) explained that he attended lectures so that he can equip himself with the skills that he needs to tackle the programming assessments during his practical sessions. In addition, Table 5.7 shows evidence of the participants taking part in

online discussions and forums to seek help on their programming assessments (ZX, FG2; CM, FG1; MJE, FG4).

5.3.2.2 Help-seeking

Table 5.7 contains the excerpts from the responses of the participants which suggested that they seek help when they were stuck in their programming assessments. The focus groups participants frequently asked for help when working on their programming assessments, particularly when they were stuck in their programming assessments. The source of help comes from their tutors, peers, the Internet, course notes, forums, and to a lesser extent, textbooks, and family members.

Asking for help from the tutors and peers appears to be the preferred source of help as the tutors and peers are familiar with the programming assessment, and are able to provide useful help to the participant's programming problem. Further, the discussion that ensued between the participants and the help provider, helped the participants understand the programming problem, and in a collegial environment. By contrast, the participants are less inclined to seek help from the Internet, course notes, forums, and textbooks as these sources of help are less useful. The participants need to search through mounds of information in order to find a solution to their programming problem. These help-seeking strategies are consistent with the instrumental help-seeking strategy that is discussed in the literature on learning theory (Chapter 3, Section 3.2.3.2).

Table 5.7: Help-seeking – evidence from responses

Responses from participants	**Source of help**
"I ask tutor a lot" (SZ, FG1)	*Tutor*
"I harass my tutor every day" (YL, FG1)	*Tutor*
"... when we're coding, I think I need more discussions with people... talk with my friends before I start it because I don't know what's going on." (ZX, FG2)	*Peers*
"... the discussion point is important for when you have troubles.. [when] stuck on something for 5 hours… you need someone else's opinion, someone else's thoughts..." (CL, FG2)	*Peers*
"I couldn't finish any assignment until the third week. Until I found some help from the tutor" (YM, FG1)	*Tutor*
"Usually, I ask MJE [peer] and Ms. A [tutor]. If I can't find a solution from either of them, I approach the Internet." (FA, FG4)	*Peer* *Tutor* *Internet*
"... lecture slides and attend the lectures, and then move on to previous workshops to see if there is something in there which I might have missed ...My sister. I'll go to her. Then I'll go to my dad." (RS, FG2)	*Course notes* *Family members*
"I use the lecture slides and I also Google lots of things." (RB, FG3)	*Course notes* *Internet*
"I try and work with other people.. hard to do it by yourself." (RB, JR & AL, FG3)	*Peers*

Responses from participants	Source of help
"We all have that mentality where we go… let's try troubleshooting this…. Nope… Miss Miss….[call the tutor]" (MJE, FG4)	*Tutor*
"..found the forums really good.. there is a tutor that is very active on there and she will jump in and explain things" (CM, FG1)	*Forum Tutor*
"I basically learned programming from [the] Internet. There are many C tutorials and everything. So, it makes my work really easy. But I also study from books. " (FA, FG4)	*Internet Textbook*
"[The] Internet really does help. On books, they can only help you so much. Books generalize it." (MJE, FG4)	*Internet*
"I went to forums and still a bit foggy. Not getting there. So, e-mail the lecturer. That's how I do it." (MJE, FG4)	*Forum Instructor*

5.3.2.3 Effort

Table 5.8 contains the excerpts from the responses of the participants which suggested that they expended effort when learning programming. The participants were engaged in a number of activities in order to learn programming and which did not appear to be mandatory to their introductory programming course. This suggests that the participants expended effort in a variety of ways so that they could learn programming. The effort expended include attempting the lab exercises, analysing programming code, setting time for revision, understanding the codes, taking notes (during lectures), and to practising programming.

Table 5.8: Effort – evidence from responses

Responses from participants	Ways to expend effort
"…there is the assignment and then there are exercises for each part of the assignment. I do all the exercises… in the lab" (MH, FG1)	*Attempt lab exercises*
"… I need to figure out what each part of it is doing...that piece results in that happening…[then]generalize it to the rest… figure out the logic..." (CL, FG2)	*Analyse programming code*
"spend like 3-5 hours per day on the weekend, just to memorize those codes and understand that code" (YL, FG1)	*Set time to revise Understand code*
"... sometimes you've just got to sit there and knock it into yourself. This is going to be important, write this bit down, or write this bit down...lecture slides on my laptop and just put in the notes below it" (RS, FG2)	*Take notes*
"After the lecture has been taught, I go home and look through the slides." (FA, FG4)	*Set time to revise*
"You have to first try it yourself first... for programming, you need to understand. I try coding myself… go online… find some questions to follow. Look for examples and follow and see the result." (LT, FG4)	*Practise programming*

5.3.2.4 Persistence

Table 5.9 contains the excerpts from the responses of the participants which suggested that they were persistent when learning programming. During the focus group discussion, participants frequently mentioned the need to be determined, frequent practise, and to keep on trying when they were learning programming. For example, participants YL and MH from Focus Group 1 (FG1) explained that practise is important in order to improve their understanding when learning programming. In addition, participant FA from Focus Group 4 (FG4) explained that programming requires determination. These findings imply that the participants should persist in learning programming despite facing obstacles.

Table 5.9: Persistence – evidence from responses

Responses from participants	Ways to persist
"..You understand it first, memorize it, you write it on paper, again and again, practise a lot. Once you practise 100 times, you remember everything. You understand everything." (YL, FG1)	*Practise*
"Determination. It's like learning a whole new language." (JR, FG3)	*Be determined*
"... very determined. You should not give up too soon." (FA, FG4)	*Be determined*
"I spent 12 days, no 14 days practising, reading the lecture notes. Making sure I understand all of them and I did all the assignments 1 to 5" (YL, FG1)	*Practise*
"You have to first try it yourself first. Programming is different. If other modules, you just have to read and memorize. But for programming, you need to understand. I try coding myself. I would go online and find some questions to follow. Look for examples and follow and see the result." (LT, FG4)	*Be resourceful. Keep trying.*
"…when there is a test coming up I also do the practises. Keep doing them… until I get them right" (MH, FG1)	*Practise and keep trying.*

5.3.3 Cognitive Engagement

Table 5.10 lists the indicators of cognitive engagement that emerged from the analysis of the focus groups:

Table 5.10: Indicators of cognitive engagement in an introductory programming course

Dimension	Indicators	
	Initial (from Literature Review)	**New (from focus groups)**
Cognitive Engagement	Deep Learning	Strategic Learning
	Surface Learning	
	Trial and Error	

The initial research model had proposed deep learning, surface learning, and trial and error as indicators of cognitive engagement, and the analysis of the focus groups confirmed the

presence of these indicators in introductory programming courses. In addition, one new indicator of cognitive engagement, *strategic learning*, emerged from the analysis of the focus groups but was eventually not included in the revised research model (Section 5.4) since only one participant had demonstrated the use of a strategic learning approach to learning programming.

5.3.3.1 Deep Learning

Table 5.11 contains the excerpts from the responses of the participants which suggested that they were using a deep approach to learning programming. Participants who appeared to engage deeply in learning programming displayed strategies such as: revising their course notes, preparing for their classes (lectures or practicals), taking notes, analysing the code, practise writing programs, understand the logic of the program, and explaining the programming concepts to their peers. The learning approaches adopted by these participants suggested that the participants were actively trying to learn programming by being involved in learning activities that can strengthen their understanding of programming. These approaches were consistent with the literature on learning theory that suggested that a deep approach to learning adds meaning to the learning task by connecting knowledge of what is known to new knowledge resulting in improved retention (Ramsden, 2003).

Table 5.11: Deep learning approaches – evidence from responses

Responses from participants	**Deep learning approach**
"After I have the initial understanding and the basis for everything, the best way for me is to teach others, explain it to others...I didn't understand it [the assignment] until I was explaining it to like 3 different people..." (CL, FG2)	*Explain to peers*
"sort of found there are 2 types of problems you can get.. the trick is to learn to read things" (CM, FG1)	*Analytical mind*
"After the lecture has been taught, I go home and look through the slides." (FA, FG4)	*Revise course notes*
"I think my preparation before the lab session is during the lecture. I will make some notes. I will try to finish everything during the lecture so that I don't have to think about it after the lecture. During the lab session, I will go back to the notes." (LT, FG4)	*Prepare for classes* *Take notes*
"Prepare doing half the assignment at least or ¾ of the assignment before" (MH, FG1)	*Prepare for classes*
"I left all the exercises until the workshop and after that, I found that each workshop was getting harder. So, after that, I always did part or half of it." (AL, FG3)	*Prepare for classes*
"I think that preparing [for practical] would be better cause then I will know where everything is." (JR, FG3)	*Prepare for classes*
"You have to first try it yourself first... you need to understand. I try coding myself. I would go online and find some questions to follow. Look for examples and follow and see the result." (LT, FG4)	*Practise programming*

Responses from participants	Deep learning approach
"Usually, I will do a lot of pre-study before the assignments. The assignment will come last, as a practise of what I have learned, and that's working." (YL, FG1)	*Revise course notes*
"you need to go away and try and think about what you need to put so you can say if this thing is selected, what happens then. I need to execute this bit of code so it produces the right result" (RS, FG2)	*Analyse code*
"..not working… find an alternative way of doing it.. then by actually seeing the difference between the 2 things you used, I normally find I actually understand why what I was doing before doesn't work" (CW, FG1)	*Analyse code*
"I am one of those (who play games)…..you see Super Mario (game) doing certain things... start wondering what logic lies behind it... what codes... and who thought about it first? So, you start wondering, let's poke something here and see what comes out. " (MJE, FG4)	*Analyse code*
"... I use previous workshops because okay I'll put this switch statement here, it works. I'll copy it into here or I'll move it into here and then I'll just fill in the empty gaps to make it relevant to this question. So it's kind of taking what you've done before and applying it to something new." (DI, FG2)	*Analyse code*
"...you need to go away and try and think about what you need to put so you can say if this thing is selected, what happens then. I need to execute this bit of code so it produces the right result" (RS, FG2)	*Understand program logic*
"...as far as logic errors go… print out the code and step through it with a pen and paper" (CM, FG3)	*Understand program logic – step through code*
"If it's a logic error, I find that I normally sort of just drill down, drill down, like if it's not doing something and there is no error, the first thing I'll do is step through it" (CW, FG1)	*Understand program logic – step through code*

5.3.3.2 Surface Learning

A surface approach to learning programming was not obvious in the focus groups. The participants were mostly aware that learning programming required understanding and not memorization of code. The following excerpts from the responses of the participants show that the participants are aware of the need to understand programming:

> *"You have to first try it yourself first. Programming is different. If other modules, you just have to read and memorize. But for programming, you need to understand. I try coding myself. I would go online and find some questions to follow. Look for examples and follow and see the result." (LT, FG4)*

> *"…the distinction… programming needs practise…. Economics and Management is memorising, and can have opinions..." (CW, FG1)*

However, the responses of three participants suggested that they used a surface approach to learning programming. Table 5.12 contains the excerpts from the responses of the participants which suggested that they were using a surface approach to learning programming. These participants appeared to be less involved with learning programming as they admitted that they did not prepare for their classes, relied only on exam tips to prepare for their programming exam, and tended to memorize programming codes. These approaches are consistent with the literature on learning theory that suggests that a surface learner tends to complete a task without giving any meaning to the task, does not attempt to reflect on the concepts or facts and their relation to prior knowledge, focuses on rote learning, and memorizes concepts (Biggs, 1987; Marton & Säljö, 1976; Ramsden, 2003).

Table 5.12: Surface learning approaches – evidence from responses

Responses from participants	**Surface learning approach**
"There's always a need to prepare [for practicals] but we tend to be lax about it." (MJE, FG4)	*Did not prepare for class*
"We just go through the slides that we studied in class at that moment [during the practical session]… " (FA, FG4)	*Did not prepare for class*
"Yes I practise but as MJE said there are certain tips given for the test. I just try to code them. I don't do anything out of the tips and I try to code the exact thing." (FA & MJE, FG4)	*Rely on exam tips*
"spend like 3-5 hours per day on the weekend, just to memorize those codes and understand that code" (YL, FG1)	*Memorize code*

5.3.3.3 Trial and Error

Table 5.13 contains the excerpts from the responses of the participants which suggested that they used a trial and error strategy to learn programming. The participants in the focus groups explained that they used a trial and error strategy to debug their programming errors particularly when the error messages were ambiguous, and when they were uncertain of the solution.

The participants attempted to debug the errors in their programming code by applying one solution at a time. When a particular solution did not work, the solution was rejected, and another solution was applied. The participants continued with this strategy until they found a solution. For example, participant JR from Focus Group 3 explained that her programming error was merely an error in naming the program. However, due to the ambiguity of the error message, participant JR adopted a trial and error approach whereby she attempted to change various parts of her programming code until she was able to figure out the solution to her error. Similarly, participant RS from Focus Group 2 (FG2) explained that her strategy to solving programming errors was to extract a solution from a previous programming problem and to apply it in the present context, while participants CW and MJE from Focus Groups 1 and 4 (FG1 & FG4) explained that using a trial and error strategy enabled them to learn from their mistakes.

In addition, the participants explained that the trial and error strategy was suitable since the solutions to programming problems differed from one solution to the other although the concept behind the programming problem was the same. For example, participant DI from Focus Group 2 (FG2) explained that he took a previous solution to a programming problem, and applied the solution by making the solution relevant to the context of the current programming assessment.

Table 5.13: Trial and Error – evidence from responses

Responses from participants	**Trial and error approach**
"… previous workshops… I'll put this switch statement here, it works. I'll copy it into here or I'll move it into here and then I'll just fill in the empty gaps to make it relevant to this question." (DI, FG2)	*Insert code that looks right*
"… go back to it and look through it [code]… really look at some code. Like, put this line before that one, change little things." (AL, FG3)	*Move code around*
"When I finally figured out it was that [why it wasn't working]; but I was changing everything else." (JR, FG3)	*Insert code that looks right*
"I go for the most unorthodox methods. I mess up the most…and slowly see where that leads to. So… I will know all the budding problems that will come out. " (MJE, FG4)	*Mess up codes to find solution*
"...I'd be like trying to extract this kind of information from this table to bring over to here....." (RS, FG2)	*Move code around*
"..find an alternative way of doing it [code].. then by actually seeing the difference between the 2 things you used, I normally find I actually understand why what I was doing before doesn't work" (CW, FG1)	*Compare codes*
"...Some of the errors … you're just like where can it be and sometimes the error statements give you a few things to do and it's like not helping." (DI, FG2)	*Code causing the error not obvious from error messages*

5.3.3.4 Strategic Learning

The literature on learning theory suggested a third learning approach - a strategic or achieving approach to learning and was discussed under the section of Learning Approaches in Chapter 2, Section 2.1.4.2. The strategic learning approach was not proposed in the initial research model as this approach did not appear to be frequently discussed in the prior literature and was deemed as an approach to studying rather than for learning (Morgan 1993).

One participant in the focus group appeared to adopt a strategic approach to learning programming. Participant SZ preferred to work on her programming assessments only if her efforts yielded the returns that she expected. Participant SZ focused on achieving success in her exams by reviewing past exam papers and ensured that the time she spent on learning

programming was proportionate to the effort that she put into learning programming. The learning approach adopted by participant SZ suggested that she had the intention of obtaining a good grade in her introductory programming course by optimising her learning and focused her efforts on learning tasks that were sufficient to achieve a good grade.

Table 5.14 contains the excerpts from the responses of participant SZ which suggested that she used a strategic approach to learning programming. However, since only one participant demonstrated the strategic learning approach, there was insufficient evidence from the focus groups to confirm that strategic learning is an indicator of cognitive engagement in this study.

Table 5.14: Strategic learning approaches – evidence from responses

Responses from participants	**Strategic learning approach**
"I want to get good grades. I make sure my code part and completion plus together over 80%" (SZ, FG1)	*Focus on expected returns*
"..for the exam, the most efficient way is you study about the previous test paper." (SZ, FG1)	*Review past papers*
"you remember all the past answers to questions and you can do your assignments because you ask the tutor and then you understand how can you do it" (SZ, FG1)	*Review past answers*
".. get 'A' cause my aim is I get good grades so I use the most efficient way to make my grade look great. But if I really know how to code, I don't know." (SZ, FG1)	*Focus on expected returns*
"...read the requirements carefully... read like the assignment page twice... I use it to make the whole structure of my code...step by step" (SZ, FG1)	*Focus on expected returns*
Time management (SZ, FG1)	*Time proportionate to effort*

5.3.4 Emotional Engagement

Table 5.15 lists the indicators of emotional engagement that emerged from the analysis of the focus groups:

Table 5.15: Indicators of emotional engagement in an introductory programming course

Dimension	**Indicators**	
	Initial (from Literature Review)	**New (from focus groups)**
Emotional Engagement	Enjoyment	Gratification
	Interest	

The initial research model had proposed enjoyment, and interest as indicators of emotional engagement, and the analysis of the focus groups confirmed the presence of these indicators in introductory programming courses. In addition, one other indicator of emotional engagement, *gratification*, emerged from the analysis of the focus groups.

5.3.4.1 Enjoyment

Table 5.16 contains the excerpts from the responses of the participants who enjoyed programming. The participants were excited when they received a new programming assessment, and when they completed their programming assessment. For example, participant RS (FG2), CM (FG1), and CL (FG2) enjoyed programming throughout their course, and participants JR (FG3), RSa (FG2), and MH (FG1) said that they were excited when they received a new programming assessment to work on.

Table 5.16: Enjoyment – evidence from responses

Responses from participants
"Programming was something I came in dreading because it is a language. Languages are not traditionally my strong suit. But finally got into it and....It's nice because in some other courses you don't get that sense of achievement along the line of doing a piece of work. " (RS, FG2)
"…thoroughly enjoy programming... thoroughly hate doing essays" (CM, FG1)
"I actually enjoy doing this subject and I want to like start on it [new programming assessment]" (JR, FG3)
"I might be the only one who is excited for another challenge [when getting a new programming assessment]" (RSa, FG2)
"…always excited to get a new program… really fun" (MH, FG1)
"Because I really enjoy programming. Maybe I go so far as to say I have fun while doing it. " (CL, FG2)
"I like programming a lot, but that's not my life" (YL, FG1)

5.3.4.2 Interest

Table 5.17 contains the excerpts from the responses of the participants who were interested in learning programming. The participants showed their interest in learning programming by using expressions such as: "programming is going to be my life" (MH, FG1), "programming is about vocation" (RS, FG1), "passionate about programming" (MJE, FG4), "programming makes me very curious" (FA, FG4) and "programming makes me wonder about the logic" (MJE, FG4).

Table 5.17: Interest – evidence from responses

Responses from participants
"…programming is going to be my life…that's what I want to be when I'm finished…" (MH, FG1)
"I feel overwhelmingly, that it's about vocation" (RS, FG2)
"I am passionate about programming." (MJE, FG4)
"I am interested because the code itself makes me very curious about what the outcome would be" (FA, FG4)
"I am one of those (who play games)…..you see Super Mario (game) doing certain things... start wondering what logic lies behind it... what codes... and who thought about it first? So, you start wondering, let's poke something here and see what comes out. " (MJE, FG4)

5.3.4.3 Gratification

During the focus groups, gratification emerged as a new indicator of emotional engagement. Table 5.18 contains the excerpts from the responses of the participants who were gratified to see the output of their program. In the literature on learning theory, research on gratification discusses "delay of gratification" and its effect on academic success. Delay of gratification is the "voluntary postponement of immediate rewards and persistence in goal-directed behaviour for the sake of later outcomes" (Mischel, Schoda, & Rodriguez, 1989, p. 933). Bembenutty (2011) explained that students who delay their gratification displayed characteristics such as a higher level of intelligence, performed better academically, and are socially more adaptable (p. 55).

However, during the focus groups, the participants felt a feeling of immediate gratification when they were able to debug the errors in their program and were able to see the output of their program. For example, participant DI from Focus Group 2 (FG2) explained that programming gave him an immediate sense of achievement compared to other courses. Participant RS from Focus Group 2 (FG2) felt rewarded each time she achieved a milestone in her programming assessment compared to a lesser sense of achievement after she had completed her assessment in the other courses. Similarly, participant CL from Focus Group 2 (FG2) felt triumphant when he was able to solve a programming problem. This finding may also link to the anecdotal evidence that was presented in Chapter 2, Section 2.1.3, whereby introductory programming course instructors had suggested that the Integrated Development Environment (IDE) that is used for writing and compiling programming code has the advantage of being able to provide novice programmers with immediate feedback on their programming errors.

Therefore, for the purpose of this study, gratification is defined as novice programmers who experience "the feeling of receiving immediate rewards typically in the form of pleasure or satisfaction as a result of hard work". The interpretation of gratification in this study focuses

on *immediate gratification* as opposed to *delay of gratification* which is a common outcome of learning in the literature on learning theory.

In addition to the research hypotheses presented in Chapter 3, Section 3.2.3, the following three hypotheses are proposed for the gratification construct.

H20: Self-efficacy beliefs in learning programming will have a positive effect on gratification.

H21a: Gratification in learning programming will have a positive effect on course grade.

H21b: Gratification in learning programming will have a positive effect on self-assessment.

Since, there is no evidence from the existing research that suggests a relationship between programming self-efficacy and gratification, and a relationship between gratification and programming performance, the hypotheses do not state a positive or negative effect.

Table 5.18: Gratification – evidence from responses

Responses from participants
"...design the user process and that is quite cool ...code for a clip and you're like that was good...with other courses I suppose there is less sense of achievement along the way...But...get to the end of an essay ...like yes it's finished. Then get to the end of a website assignment ...still have that yes I'm finished but then it's like, all those yeses in between are nice as well. " (RS, FG2)
"...especially if I figure it out for myself that's a really rewarding feeling." (RS, FG2)
"I think it's easier for me. Once the code is working, it's like you achieve something." (AL, FG3)
"..You're getting somewhere while doing it [programming]. Sometimes you are writing 5 words [in courses other than programming] and you think do I agree with it. Do I think it's right?" (DI, FG2)
"..Just kind of smile thinking you've done it. It's finished." (DI, FG2)
"…started with only like a blank piece of paper basically, and I put everything into it so it is able to come out with whatever comes out at the end..." (MH, FG1)
"…. I enjoy seeing it grow from a blank, empty file to presenting this thing on the screen and having it do my bidding so to speak. It's just a very satisfying feeling.." (CM, FG1)
"…very satisfying feeling when it works… it doesn't, it's very frustrating…" (CM, FG1)
"Triumph [when I find a solution to my programming problem]" (CL, FG2)
"Not frustration but definitely excitement when I finish [my programming assessment]." (RB, FG3)

5.4 Revised Research model with Hypotheses

Figure 5.1 shows the revised research model with the hypotheses after confirming the indicators of engagement from the focus groups. Gratification is one new indicator of emotional engagement and the relationships have been added into the research model.

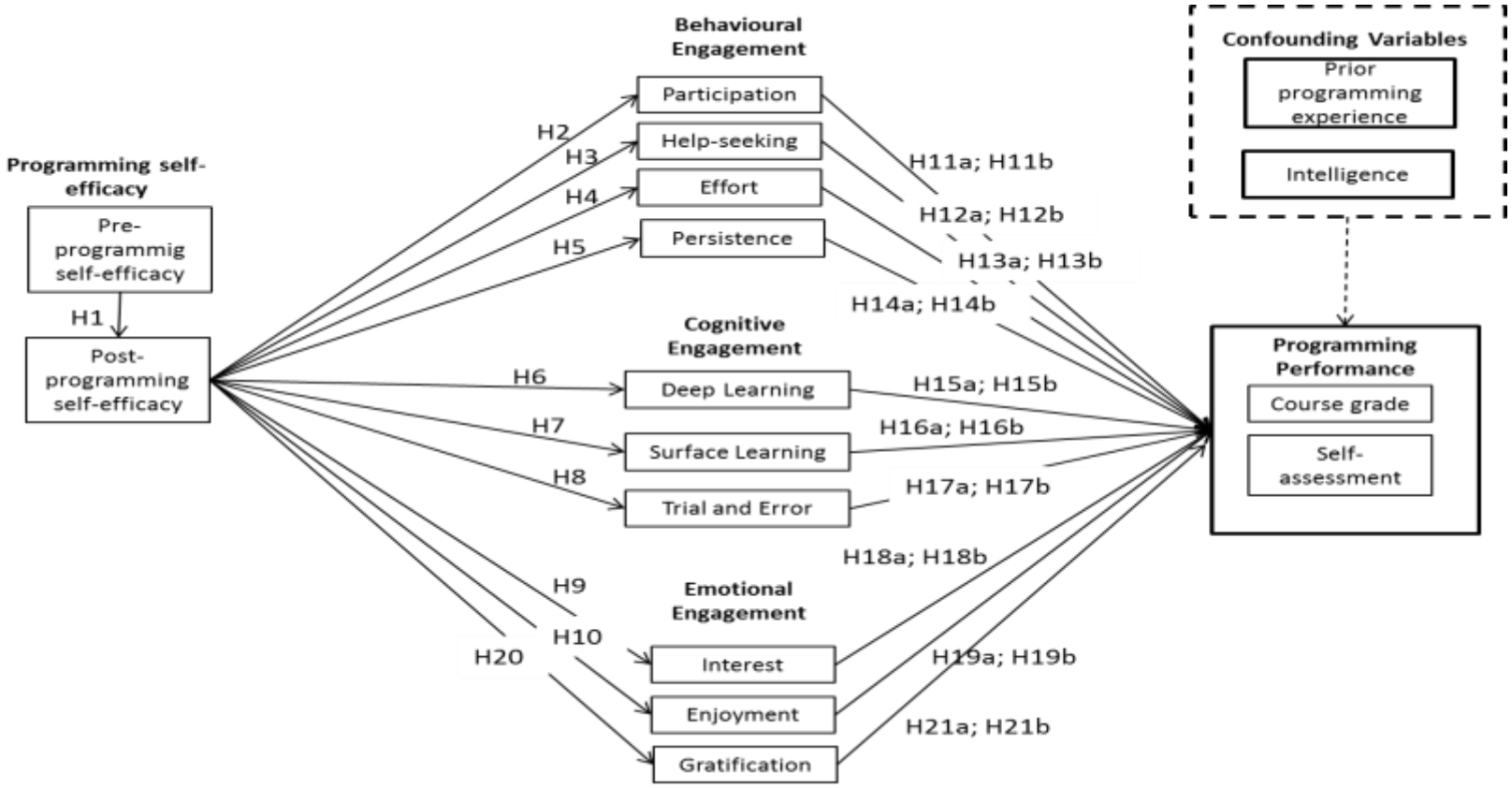


Figure 5.1: Revised research model with hypotheses

5.5 How learning programming differs from learning in other courses?

Although not directly related to the main objective of the focus groups, the participants in the focus groups made clear distinctions between their introductory programming courses and the other courses that they were enrolled in. Table 5.19 provides excerpts from the responses of the participants and shows the differences between learning programming and learning in other courses.

The participants found that their introductory programming course required a hands-on approach to learning compared to other courses which required reading, memorization, and opinions. The participants also found that they were able to assess their programming performance since programming assessments were either right or wrong compared to other courses which were based on analysis and arguments. A third distinction that the participants made was related to the understanding and the application of concepts. Although the participants understood the programming concepts, they had to apply the concepts to the context of the programming problem. However, in other courses, they found that applying concepts to a problem was relatively straightforward. The fourth distinction that the participants made was related to the type of help that the participants sought. In programming, the participants said that they could seek help from peers to solve the errors in their program design and their programming codes while in other courses they found that they were able to discuss their assessments without asking for specific solutions to the problem. The final distinction that the participants made was in the viewing of their results. In programming, one participant explained that she could see the result of her programming assessment only after she had corrected the errors in her program and was able to see the output of the program. By contrast, the output of the assessments in other courses was visible immediately, as the participant worked on producing the solution to the assessment. These findings provide strong evidence that programming requires skills, and learning approaches that are different from other courses.

Table 5.19: Comparison between introductory programming course and other courses

How they differ	Introductory Programming	Other Courses
Hands-on vs. reading, memorization, opinion	*"Need to understand and try coding on your own" (LT, FG4)* *"Practise" (CW, FG1)* *"… got to go at it... the trick to learning is doing… it's a very practical subject" (CM, FG1)* *"… more of a hands on approach.." (DI, FG2)*	*"Reading and memorization is sufficient" (LT, FG4)* *"Economics and Management is memorize, and can have opinions" (CW, FG1)*
Right or wrong answer vs. Analysis and argument	*"If it doesn't work, you're not going to get the marks" (JR & RB, FG3)* *"…either right or it's wrong. Or you're halfway there" (MH, FG1)* *"…you do it in the right way and you get the mark" (ZX, FG2)* *"…it's either right or you're wrong…. Very black and white… the exam either works or doesn't" (CW, FG1)* *"…. Either right or it's wrong so you have to practise it until you get it right…" (MH, FG1)* *"…with code you have to agree with it because it's right or wrong...going for the same endpoint… the right answer" (DI, FG2)* *"… different ways of doing it…. but they are all going to be doing the same thing" (RS, FG2)*	*"Wish for the best and hope that it is right" (JR, FG3)* *"An English essay… there's a continuum" (CW, FG1)* *"On an essay, you are trying to back up a statement that you have made… different opinions… going to different end points but can interlace all of your ideas and opinions to further advance your opinion" (DI, FG2)* *"Can have opposite opinions" (RS, FG2)*
Customise application of concepts vs. Straightforward application of concepts	*"You get it once, it evolves! Understanding it may not mean that you can do it again the next time" (MJE, FG4)* *"Programming questions are tricky and you cannot always apply the same method. It's always different." (FA, FG4)* *"You study programming and you implement it. It's two different things." (MJE, FG4)*	*"You get it once, you get the whole thing" (MJE, FG4)* *"In Maths, if you know the method, if your concepts are clear, you can solve any question with any numbers" (FA, FG4)* *"Study and you can get it" (MJE, FG4)*

Help to solve problem vs. discussion	*"Need more discussion with friends before I start" (ZX, FG2)* *"Discussion… can get help with problem… help each other with code or design" (DI, FG2)*	*"Have a look at the textbook and see if there is a resource and then talk to people" (ZX, FG2)* *"For an essay, you would discuss it but you wouldn't help each other write it… that would be borderlining on plagiarism" (DI, FG2)*
Result comes with success vs. Results at all times	*"…coding is not a result. It's just a pathway to your result… you can't see the result until you run the program… hard part is that you can't see the things you want…" (SZ, FG1)*	*"… in design, you can see what you drew" (SZ, FG1)* *"… in maths, you can get the result, numbers you can see…" (SZ, FG1)* *"on a spectrum… Math at one end and the fuzzy sort of management, economics… at the other end. I would say there is a bit of method in it. Then put programming in the middle of that" (CW, FG1)*

5.6 Chapter Summary

The focus groups were analysed using an inductive and a deductive approach, and the analysis conforms to the *key concepts* analysis framework that was proposed by Krueger and Casey (2009). The focus group participants provided rich narratives of their engagement during their introductory programming course. The findings of the focus groups confirmed that *persistence*, *help-seeking*, *effort* and *participation* are indicators of behavioural engagement, *deep learning*, *surface learning* and *trial and error* are indicators of cognitive engagement, while *enjoyment* and *interest* are indicators of emotional engagement. *Gratification* emerged as a new indicator of emotional engagement and was defined within the context of an introductory programming course. Hypotheses were then developed for gratification.

The Phase 3 survey questionnaire could then be developed to examine the relationship between programming self-efficacy and each of the indicators of engagement, and the relationship between the indicators of engagement and the programming performance of novice programmers. The development of the survey questionnaire is discussed in the following chapter.

Finally, the findings from the focus groups revealed five ways in which learning programming was different from learning in other courses. These findings support the discussion in Chapter 2, Section 2.1.3 that distinguishes programming courses from other courses and argues why novice programmers find programming difficult.

Chapter 6: Survey Design

The development of the questionnaire in Phase 1 and Phase 3 of this study followed a three-stage instrument development process that includes item creation, scale development, and testing the questionnaire (Moore & Benbasat, 1991). The data collection for this study and the preparation of the data for analysis are also discussed in this chapter. Appendix H, Table AH.1 provides a summary of the reliability and validity assessments that were performed on the measurement items in this study.

6.1 Phase 1: Survey Questionnaire

The objective of the Phase 1 survey questionnaire was to collect data to partially answer Research Question 1 (RQ1): "*What is the effect of self-efficacy on the novice programmer's engagement in an introductory programming course?*"

In this phase, the data for the pre-programming self-efficacy beliefs of the participants was collected using a survey questionnaire. In addition, the survey collected the demographic data of the participants, and data for the two confounding variables (prior programming experience and intelligence) in this study.

6.1.1 Instrument Development

The first step in the instrument development process was to create a pool of items that may be used to measure the constructs in the research. Then the content validity of the items was assessed by obtaining the opinion of a group of experts. Appendix C contains a copy of the Phase 1 survey questionnaire.

6.1.1.1 Item Creation for the Programming Self-Efficacy construct

The items to measure the programming self-efficacy construct were developed from two sources. They are: from existing empirical research on programming self-efficacy and from examining the learning objectives of several introductory programming courses. These sources of item development adhere to the recommendations made by MacKenzie et al. (2011) to develop items that represent a construct. The proposed items to measure the programming self-efficacy of the participants at the beginning (and at the end) of the introductory programming course were adapted from the programming self-efficacy scale that was developed and validated by Ramalingam and Wiedenbeck (1998). Ramalingam and Wiedenbeck developed the items in the programming self-efficacy scale by reviewing other self-efficacy scales from the fields of reading/writing, mathematics, using computer software, air traffic control, and the self-efficacy scale by Compeau and Higgins (1995). The scale was also developed in consultation with three experts on self-efficacy theory and by three teachers who were teaching the C++ programming language (Ramalingam & Wiedenbeck, 1998). The programming self-efficacy scale was also developed based on Bandura's (1986) three dimensions of self-efficacy: *magnitude*, *strength*, and *generality*.

Ramalingam and Wiedenbeck's programming self-efficacy scale consists of four factors and 32 items that are specific to the C++ programming language. The four factors include *independence and persistence*, *complex programming tasks*, *self-regulation*, and *simple programming tasks* (Ramalingam & Wiedenbeck, 1998). After reviewing the currency of the items in the scale, and examining the suitability of the items on all types of programming languages, 20 out of the 32 items were selected for use in the programming self-efficacy scale in this study. Two items from the original scale were merged into item B/APSE10 (Table 6.1) since the two items on the original scale were statements on using resources to learn programming.

Table 6.1 presents the initial pool of items for the programming self-efficacy scale in this study. These items were used in Phase 1 of the survey questionnaire to determine the pre-programming self-efficacy beliefs of the participants at the beginning of the course, and in Phase 3 of the survey questionnaire to determine the post-programming self-efficacy beliefs of the participants at the end of the course. In total, 5 factors consisting of 25 items were identified to measure the programming self-efficacy of the students. The five factors include the four existing factors from Ramalingam and Wiedenbeck's (1998) programming self-efficacy scale, and a new factor – *general self-efficacy* – consisting of 6 items. The new factor asks the students to rate their overall confidence in performing well in the introductory programming course. The items for the general self-efficacy scale were borrowed from the existing literature on academic self-efficacy.

All items start with the statement "*I am confident that….".*To distinguish between the pre- and post-programming self-efficacy items, the item code "BPSE#" was used to label the items that were administered in Phase 1 of the survey questionnaire, and the item code "APSE#" was used to label the items that were administered in Phase 3 of the survey questionnaire.

Table 6.1: Pool of items to measure programming self-efficacy

Construct:		**Programming self-efficacy**	
Factor	**Code**	**Item**	**Reference**
Note: All items start with *"I am confident that……."*			
General self-efficacy	**B/APSE1**	I can complete a new task that is assigned to me if I keep trying.	**Bresó et al. (2011)**
	B/APSE2	I can stick to completing a task even though it may be unpleasant.	**Sherer et al. (1982)**
	B/APSE3	When I learn something new, I will not give up easily if I am not successful initially.	
	B/APSE4	When unexpected problems occur I can handle them well.	

Construct:		**Programming self-efficacy**	
Factor	**Code**	**Item**	**Reference**
Note: All items start with *"I am confident that……."*			
	B/APSE5	I can achieve the goals that I set for myself in this programming course.	**Bresó et al. (2011)**
	B/APSE6	I have the capability to learn the contents of this programming course.	**Walker et al. (2006)**
Independence and persistence	**B/APSE7**	I can complete my programming project if I had a lot of time.	**Ramalingam & Wiedenbeck (1998)**
	B/APSE8	I can complete my programming project once someone else helps me get started.	
	B/APSE9	I can complete my programming project if I can call someone for help if I get stuck.	
	B/APSE10	I can complete my programming project if I refer to resources such as the built-in help, programming reference manuals, or online programming forums.	
	B/APSE11	I can complete my programming project if someone showed me how to solve the problem first.	
	B/APSE12	I can find ways of overcoming the problem if I get stuck at a point while working on a programming project.	
	B/APSE13	I can correct (debug) all the errors in a long and complex program that I have written, and make it work.	
Complex programming tasks	**B/APSE14**	I can identify the objects in the problem domain and declare, define, and use them.	**Ramalingam & Wiedenbeck (1998)**
	B/APSE15	I can understand and apply the fundamental object-oriented programming concepts used in my programming course.	
	B/APSE16	I can make use of a pre-written library in the programming integrated development environment (IDE).	
	B/APSE17	I can write a program to solve any given problem as long as the specifications are clear.	
	B/APSE18	I can mentally trace through the execution of a complex program given to me.	

Construct:		**Programming self-efficacy**	
Factor	**Code**	**Item**	**Reference**
Note: All items start with *"I am confident that……."*			
Self-regulation	**B/APSE19**	I can find a way to concentrate on my programming project, even when there are many distractions around me.	**Ramalingam & Wiedenbeck (1998)**
	B/APSE20	I can find ways of motivating myself to program, even if the problem area was of no interest to me.	
Simple programming tasks	**B/APSE21**	I can write programming code that runs without errors (no syntax errors)	**Ramalingam & Wiedenbeck (1998)**
	B/APSE22	I can write programming code that is logical.	
	B/APSE23	I can write a small program for a small problem that is familiar to me.	
	B/APSE24	I can understand the language structure and the usage of the reserved words /keywords in the programming language.	
	B/APSE25	I can write a reasonably sized program that can solve a problem that is only vaguely familiar to me.	

6.1.1.2 Expert Review

The items to measure programming self-efficacy were assessed for content validity in two stages in order to ensure that the items were accurately measuring the construct (Field, 2013). In the first stage, an academic examined the items to determine their suitability and currency. Since there were no alternative programming self-efficacy scales that were developed subsequent to Ramalingam and Wiedenbeck's (1998) programming self-efficacy scale, the programming self-efficacy scale was compared with recent general academic self-efficacy scales that were proposed by Bresó et al. (2011) and Walker et al. (2006). The academic self-efficacy scale items measure general self-efficacy beliefs of students compared to the programming self-efficacy scale developed by Ramalingam and Wiedenbeck (1998). Table 6.2 discusses the academic self-efficacy scales in comparison to the programming self-efficacy scale developed by Ramalingam and Wiedenbeck (1998). With the exception of one scale item by Bresó et al. (2011), all other items from the academic self-efficacy scale were similar to the *independence and persistence* factor in the programming self-efficacy scale, leading to the decision that the pool of items in the programming self-efficacy scale is current.

Table 6.2: Comparison between general academic self-efficacy scales items and programming self-efficacy scale items.

Author	Academic Self-Efficacy Items	Discussion
Bresó et al. (2011)	I will be able to properly use the practical skills I learned over the last academic year.	Not appropriate in this context as this study is about an introductory programming course. There may be no basis for comparison for the participants.
	I will be able to understand the most difficult subjects this academic year.	Addressed in *independence and persistence* factor items of the programming self-efficacy scale
	If I try hard enough, I will be able to complete every task in class.	
	I will be able to learn in class, even the most complicated concepts [2].	
	If I try hard, I will be able to do the most difficult tasks related to my studies.	
Walker et al. (2006)	I am certain I can learn the ideas and skills taught in the class.	Similar to the item marked [2] and addressed in the *independence and persistence* factor.

The second stage examined the appropriateness of the items in the programming self-efficacy scale. The programming self-efficacy scale was reviewed with four introductory programming course instructors in New Zealand, and one introductory programming course instructor in Malaysia. Table 6.3 outlines the feedback and the decisions that were made regarding the appropriateness of the items in the programming self-efficacy scale. Once the expert review was completed, 25 items were identified to measure the programming self-efficacy construct. The survey instrument was developed and the items to measure programming self-efficacy used a 5-point Likert scale ranging from 1 (not confident at all) to 5 (very confident).

Table 6.3: Suggestions for improvement from expert reviewers

No.	Suggestions to improve	Response
1.	Use of an appropriate term that will encompass all the assessments in the programming course – programming project may refer to only one type of assessment in the course.	The term *"programming assessment"* was used instead of *"project"* and the terminology was explained at the beginning of the questionnaire in Phase 1 and Phase 3 (Appendix C & D). The terminology is also explained in Chapter 1, Section 1.4 in this thesis.

2.	Item B/APSE13 – the confidence to debug should apply to a program of any size and complexity.	B/APSE13 amended to: *"I can correct (debug) all the errors in a program that I have written, and make it work."*
3.	Item B/APSE14 - students may not know how to implement the object-oriented concepts, but they will be familiar with concepts such as class, object, and inheritance.	B/APSE14 amended to: *"I can apply the right programming concepts to solve a problem given to me."*
4.	Customise the questionnaire based on the programming language used in the course so that the participants are able to relate to the questions better, and the meaning of the questions is not compromised.	Questions were customised to the programming language that was used in the Higher Educational Institution (HEI) where the questionnaire was administered. For example, a course which uses the Java programming language might have a customised question which reads *"I am confident that I can complete my Java programming assessment if I could call someone for help if I got stuck." – BPSE9* or *"I can make use of the ECS100 pre-written library in the programming integrated development environment." – BPSE16.*
5.	Change present tense to past tense when the questionnaire is administered again in Phase 3, since the students will be assessing their self-efficacy beliefs once they have completed their course.	The Phase 3 questions ask novice programmers to reflect on their progress during the course and to assess their current level of confidence in tackling new tasks and learning programming. Thus, the items will remain in the present tense.

6.1.1.3 Item Creation for other questions in Phase 1 survey

Although not explicitly required in the research question, the demographic data of the participants were nevertheless obtained in order to understand the profile of the participants. The age, gender, ethnicity, and type of major data was collected. The age data was an open-ended question in the questionnaire, but was later grouped into two categories of "*25 years and less*", and "*above 25 years*" during the data preparation stage. The gender data used a dichotomous categorical scale of *male* and *female* while the ethnicity data was collected using a categorical scale ranging from *New Zealand European*, *New Zealander*, *Other European*, *Pasifika*[5], *Maori*, *Asian*, *Middle Eastern*, *Latin American*, *African* and *Others*. The categories for the type of major were customised to the programmes that were offered at each participating institution. The majors were then grouped into two categories ("*Information Technology (IT) major"*, and "*non- Information Technology (IT) major"*) during the data preparation stage.

[5] *Pasifika is a term used by the New Zealand government agencies to refer to the indigenous people of the Pacific Islands (which includes Polynesia, Melanesia and Micronesia).*

Next, items were created to measure prior programming experience, which is a confounding variable in this study. Participants were asked if they had prior programming experience, and how they had gained their programming experience. The participants were also asked to rate their level of programming experience on a scale of 1 to 5, with 1 being *limited programming experience* and 5 being *extensive programming experience*.

6.1.1.4 Other data collected

As previously proposed in Chapter 3, Section 3.3, the school result of the participant was used to measure intelligence and is a confounding variable in this study. The school result of the participant serves as an entry qualification into the HEI. The data for the school result was obtained from the institution's students records after obtaining approval from the participant. A copy of the Participant Consent Form that asks for permission to access their school results is attached in Appendix E. The signed Participant Consent Form was sent to the Manager responsible for student records as evidence of consent to access the participant's school results. The school results were obtained 6 weeks after the end of the introductory programming course. The school results were coded using a scale of 1 to 5 as illustrated in Table 6.4.

Table 6.4: Grades used for school results (Intelligence confounding variable)

Scale	Result		
	Grade	**Scores**	**Classification**
1	E	< 30%	Fail
2	D	30% - 49%	Poor
3	C	50% - 64%	Pass
4	B	65% - 79%	Merit
5	A	80% - 100%	Distinction

Since the participants were of various nationalities, there were differences in the grading scale used in the school results. To ensure consistency in the interpretation of the school results, the UCAS 2015 International Qualifications (UCAS, 2014) reference guide was used to map the school results to the grading scale used in this study.

6.1.2 Scale Development

According to Moore and Benbasat (1991, p. 199), the objective of the scale development stage is to assess construct validity and to identify items that lack clarity. Since the programming self-efficacy scale had been validated in previous research, efforts were focused instead on validating the engagement and self-assessment scales which are discussed in Chapter 6, Section 6.2.2.

6.2 Phase 3: Survey Questionnaire

The objective of the Phase 3 survey questionnaire was to collect data to answer Research Question 1 (RQ1) – "*What is the effect of self-efficacy on the novice programmer's engagement in an introductory programming course?*", and Research Question 2 (RQ2) – "*What is the effect of the novice programmer's engagement on their performance in an introductory programming course?*" To answer RQ1 and RQ2, the following data was collected:

a) post-programming self-efficacy scores of the participants;
b) engagement scores (behavioural engagement, cognitive engagement, and emotional engagement) of the participants; and
c) the participant's self-assessment of their performance in their introductory programming course.

In addition to the survey questionnaire, the following data was collected separately to answer RQ1 and RQ2:

a) the participant's programming grade; and
b) the school results of the participant (which was used to measure the confounding variable – intelligence and is discussed in Section 6.1.1.4).

6.2.1 Instrument Development

The first step in the instrument development process for the Phase 3 survey questionnaire started with the creation of a pool of items to measure the constructs in the research. Then, to assess the content validity of the items, the opinion of a group of experts was sought, followed by two rounds of card sorting. Appendix D contains a copy of the Phase 3 survey questionnaire.

6.2.1.1 Item Creation for the Post-programming Self-Efficacy Construct

The post-programming self-efficacy scale from Phase 1 was repeated in Phase 3 of the survey questionnaire. The participants were asked to reflect on their progress during their introductory programming course, and to answer the questions based on their current level of confidence in tackling new tasks, and learning programming. The reason for doing so is to determine if there were changes in the self-efficacy beliefs of the participants (see H1, Chapter 3, Section 3.2.2). Similar to Phase 1, a 5-point Likert scale was used, ranging from 1 (not confident at all) to 5 (very confident).

6.2.1.2 Item Creation for the Engagement Construct

The items for the indicators of the engagement construct contained a mix of previously validated items and new items. The items were developed from the literature on learning theory, from the Phase 2 focus groups, and from the conceptual definition of the constructs. Morgan (1997) suggested that item wordings may be derived from focus groups and can be an effective way for the researcher to convey his/her intent to the respondent (p. 25). In

addition, using item wordings from focus groups can allow for constructs to be identified and to determine the dimensions in which these constructs may exist (Morgan, 1997). Items that were borrowed from the literature on learning theory were re-worded to make them relevant to the context of learning programming.

Tables 6.6 – 6.9 contains the items that measure the indicators of the behavioural engagement construct. Tables 6.11 – 6.13 contains the items that measure the indicators of the cognitive engagement construct while Tables 6.15 - 6.17 contains the items that measure the indicators of the emotional engagement construct. Since the research model has been proposed as a reflective-formative type (Chapter 3, Section 3.4), a global item indicator which is measured reflectively is required to assess the convergent validity of the higher-order construct (Hair et al., 2014). Thus, Tables 6.5, 6.10, and 6.14 contain the global item(s) for the higher-order behavioural engagement, cognitive engagement, and emotional engagement constructs. The item codes that are marked with a (*) are negatively worded items. DeVellis (2012) explains that negatively worded items are useful in a survey questionnaire in order to "avoid the respondent's tendency to agree with an item irrespective of their content" (p. 83). All engagement items used a 5 point Likert scale, with 1 being "Strongly Disagree", and 5 being "Strongly Agree".

Behavioural Engagement

Table 6.5: Global items to measure the higher-order behavioural engagement construct

Construct:	**Behavioural Engagement**
Code	**Item**
BEGI1	I was always involved in learning programming throughout the course.
BEGI2	I was constantly working on my programming assessments during the course.

Table 6.6: Items to measure participation

Construct:	**Participation**	
Code	**Item**	**Reference**
BEPA1	I worked all the way through the course to complete all of my programming assessments.	**Definition**
BEPA2	Is attendance compulsory for your: a) Lectures? (Y/N); and b) Practicals? (Y/N) If yes, please state the percentage of attendance at your: a) Lectures (%); and b) Practicals (%)	**Focus Groups**
BEPA3	Outside of my class time, I often continue discussing my programming assessments with my friends, and/or family members.	**Focus Groups**
BEPA4	I often volunteer to answer questions that my peers ask about programming on discussion boards and/or during class time.	**Focus Groups** **Shaw (2013)**

In Chapter 3, Section 3.2.3.2, instrumental help-seeking was proposed as the desired type of help-seeking strategy in this study, while help-seeking threat and help-seeking avoidance were strategies to be avoided. The findings from the focus groups established that the novice programmers generally used an instrumental help-seeking strategy when they were stuck in their programming assessments. Thus, items BEHS1, BEHS3, and BEHS5 measure instrumental help-seeking strategies, while item BEHS2* measures help-seeking avoidance, and BEHS4* measures help-seeking threat.

Table 6.7: Items to measure help-seeking

Construct:	**Help-seeking**	
Code	**Item**	**Reference**
BEHS1	Asking for help would be one of the first things I would do if I had errors in my programming assessment.	**Karabenick (2003)**
BEHS2*	I do not ask for help when I am stuck in my programming assessment.	**Karabenick (2003)**
BEHS3	I get help whenever I can to get through my programming assessments.	**Karabenick (2003)** **Focus Groups**
BEHS4*	I would feel like a failure if I asked for help with my programming assessment.	**Karabenick (2003)** **Focus Groups**
BEHS5	I expect to be given a hint to the solution when I ask for help to debug my programming errors.	**Focus Groups**

Table 6.8: Items to measure effort

Construct:	**Effort**	
Code	**Item**	**Reference**
BEEF1	I spend more time working on my programming assessment compared to assessments in other courses.	**Focus Groups**
BEEF2	I try to stay on top of my programming assessments by starting and completing them early.	**Focus Groups** **Finn & Zimmer (2012)**
BEEF3	When I don't understand a programming concept, I take time to review my lecture notes and online resources that would help me understand the concept.	**Focus Groups** **Finn & Zimmer (2012)**
BEEF4*	I think it's smart to complete my programming assessments by reviewing the program that my peers have coded.	**Focus Groups**
BEEF5	I often practise writing programs in order to understand a programming concept.	**Focus Groups**
BEEF6	I organise my time effectively in order to maximise the effort spent on my programming assessment.	**Focus Groups**

Table 6.9: Items to measure persistence

Construct:	**Persistence**	
Code	**Item**	**Reference**
BEPE1	I can spend hours debugging an error in my programming assessment without feeling like giving up.	**Focus Groups**
BEPE2*	I give up debugging an error in my programming assessment if I don't get it right after 2 or 3 attempts.	**Focus Groups**
BEPE3	I review my lecture notes and refer to other resources again and again in order to try and understand a difficult programming concept or problem.	**Focus Groups**
BEPE4	When I don't understand a programming problem, I keep practising until I understand the problem.	**Focus Groups**
BEPE5	When my program doesn't work, I am determined to fix the errors no matter how long it takes.	**Focus Groups**

Cognitive Engagement

Table 6.10: Global items to measure the higher-order cognitive engagement construct

Construct:	**Cognitive Engagement**
Code	**Item**
CEGI1	I carefully think through the programming concepts that I use to solve my programming problems.
CEGI2	I use various strategies to learn programming throughout the course.

Table 6.11: Items to measure deep learning

Construct:	**Deep Learning**	
Code	**Item**	**Reference**
CEDL1	When I am given a programming problem to solve, I try to understand the program logic before I start coding the solution.	**Focus Groups**
CEDL2	I carefully read through the assessment criteria for my programming assessment so that I know what I have to do in order to achieve a particular grade.	**Focus Groups**
CEDL3	I find that programming problems often stir my curiosity and make me think deeply about how to code the solutions to the problems.	**Focus Groups**
CEDL4	When I encounter an error in my programming assessment, I try to work out why my piece of code didn't work by stepping through the logic of the code that I had written.	**Focus Groups**

Table 6.12: Items to measure surface learning

Construct:	**Surface Learning**	
Code	**Item**	**Reference**
CESL1	I try to memorize the steps for solving programming problems presented in the lecture slides or from online resources.	**Miller et al. (1996)** **Focus Groups**
CESL2	When working on my programming assessment, I re-use the code from my course notes or from other resources although I am not sure how the code works.	**Carbone, Hurst, Mitchell, & Gunstone (2009)** **Focus Groups**
CESL3	Once I get help with solving a programming problem, I move on and do not wonder why the error occurred.	**Focus Groups**
CESL4	When I encounter an error in my programming assessment, I try to debug the error by inserting any code which looks right to me.	**Focus Groups**
CESL5	I find that I often rely on my friend's suggestions to make my programming code work.	**Carbone et al. (2009)**
CESL6	When I encounter an error in my programming assessment, I fix the error without actually understanding where my program went wrong.	**Carbone et al. (2009)**
CESL7	I find reviewing previously solved programming problems a good way to prepare for my programming assessments.	**Miller et al. (1996)** **Focus Groups**
CESL8	I study only the materials that are required to obtain a good grade in my programming course.	**Focus Groups**

Since trial and error was proposed as a new indicator of cognitive engagement and has not been empirically tested previously, the development of the items to measure trial and error was mainly from the focus groups.

Table 6.13: Items to measure trial and error

Construct:	**Trial and Error**	
Code	**Item**	**Reference**
CETE1	If I get a logic error in my programming assessment, I debug my code by stepping through the code line by line.	**Focus Groups**
CETE2	I try different solutions when I work on practise problems.	**Focus Groups**
CETE3	I often try different ways to solve my programming problem.	**Focus Groups**
CETE4	I find that I learn from my mistakes when I try different ways to solve my programming problems.	**Focus Groups**
CETE5	When I work on a programming problem, I move the codes around to make it work.	**Focus Groups**

Emotional Engagement

Table 6.14: Global items to measure the higher-order emotional engagement construct

Construct:	**Emotional Engagement**
Code	**Item**
EEGI1	Programming stirs up positive emotions in me.

Table 6.15: Items to measure interest

Construct:	**Interest**	
Code	**Item**	**Reference**
EEIN1	My programming assessments take priority over the assessments from other courses.	**Focus Groups**
EEIN2	I would enroll in another programming course if I had the opportunity to do so.	**Focus Groups**
EEIN3	I am passionate about programming.	**Focus Groups**
EEIN4	My programming course suits me better than the other courses that I am currently enrolled in.	**Focus Groups**
EEIN5*	I tend to delay working on my programming assessments.	**Focus Groups**

Since gratification was proposed as a new indicator of emotional engagement and has not been empirically tested previously, the development of the items to measure gratification was mainly from the focus groups.

Table 6.16: Items to measure gratification

Construct:	**Gratification**	
Code	**Item**	**Reference**
EEGR1	I feel a deep sense of satisfaction when I finally get my program to work.	**Focus Groups**
EEGR2	I feel relieved when I complete my programming assessment.	**Focus Groups**
EEGR3	Learning programming is a rewarding experience.	**Focus Groups**
EEGR4	I feel a strong sense of achievement when I complete my programming assessments compared to assessments in other courses.	**Focus Groups**
EEGR5	Once I complete my programming assessment, I feel pleased that I have successfully completed a challenging piece of assessment.	**Focus Groups**

Table 6.17: Items to measure enjoyment

Construct:	**Enjoyment**	
Code	**Item**	**Reference**
EEEN1	My programming course is stimulating compared to the other courses that I am enrolled in.	**Bishop-Clark et al. (2007)** **Focus Groups**
EEEN2	I like writing programs.	**Bishop-Clark et al. (2007)**
EEEN3	The challenge of coding solutions to programming problems appeals to me.	**Bishop-Clark et al. (2007)**
EEEN4*	I tend to feel frustrated when I am coding solutions to programming problems.	**Focus Groups**

6.2.1.3 Item Creation for the Programming Performance Construct

Programming performance was measured using two constructs: self-assessment of performance, and programming grade.

Self-assessment

The wordings for the items to measure self-assessment were derived from the existing literature on learning theory and based on the definition of self-assessment. Table 6.18 contains the items that measure the self-assessment construct, and where applicable, their references. Item codes marked with a (*) are negatively worded items. Like the engagement items, the self-assessment items used a scale of 1 to 5 with 1 being "Strongly Disagree" and 5 being "Strongly Agree".

Table 6.18: Items to measure self-assessment

Construct:	**Self-assessment**	
Code	**Indicator**	**Reference**
SA1	I believe that I have performed well in this programming course.	**New**
SA2	I believe that I have learned adequate programming skills in this course.	**New**
SA3*	I don't think that the amount of time I put into my programming course will be reflected in my final grade.	**New**
SA4*	I don't think that the amount of effort I put into my programming course will be reflected in my final grade.	**New**
SA5	I now understand the bigger picture of what programming is and what you can do with programming.	**Bishop-Clark et al. (2007)**

Programming Grade

The programming grade construct was measured by a single item (ProgGrade). Although a single-item measure for a construct is supported by SEM (Ringle, Sarstedt, & Straub, 2012), there are advantages and disadvantages in the use of single-item measures. According to Straub, Boudreau, & Gefen (2004), single-item measures may be used if there are no alternatives. Chapter 3, Section 3.2.1 discusses why a single-item was used to measure programming grade.

The programming grade data was obtained from the institution's students records after obtaining approval from the participants. This was necessary to ensure the integrity of the data. A copy of the Participant Consent Form that asks for permission to access the participant's programming grade is attached in Appendix E. The signed Participant Consent Form was sent to the Manager responsible for student records as evidence of consent to access the participant's programming grade. The programming grades were obtained between 6 and 10 weeks after the novice programmers completed their assessments.

The category of grades based on the marks obtained by the participants in HEIs in Malaysia or in New Zealand were similar. Table 6.19 shows the breakdown of marks and grading scale used in both countries, and the scaled grades that were used for the analysis in this study.

Table 6.19: Breakdown of marks and grades in New Zealand and Malaysia, and scaled grades used

	New Zealand	**Malaysia**	**Scaled Grades**
Marks (%)	**Grade**	**Grade**	**Grade**
90-100	A+		
85-89	A	A+	A
80-84	A-		
75-79	B+	A	
70-74	B	B+	B
65-69	B-	B	
60-64	C+	C+	
55-59	C	C	C
50-54	C-	C-	
40-49	D	D	D
30-39	E	F+	E
<30		F	

6.2.1.4 Expert Review

Once a pool of items was developed for the engagement and self-assessment constructs, face validity of the items was examined. Face validity refers to how well an item fits the construct's definition (Zikmund, Babin, Carr, & Griffin, 2010). The items were subjected to a review by an expert panel of three academics with experience in teaching introductory

programming courses. The academics were briefed on the objective of the research, given a list of construct definitions, and the pool of items for each construct.

All three academics questioned the use of attendance (BEPA2) to measure participation and unanimously agreed that the attendance to lectures and practical/laboratory sessions may not be mandatory in all introductory programming courses. In addition, one academic commented that *"Even if a student had a 100% attendance, we will know that the student was physically present in class. But how will we know that the student was actually engaged in the lectures?"* Thus, although the literature on learning theory suggests that attendance in class is one way of measuring participation, attendance may not be a reliable measure for participation in introductory programming courses. Subsequently, item BEPA2 was dropped from the pool of items that measure participation.

The expert panel review had a similar concern on item BEPA4. According to two of the academics, although answering questions on a discussion board is a measure of participation, an assumption should not be made that the student is not participative if the student does not answer questions on a discussion board. Thus, as a result of the feedback from the two experts, BEPA4 was deleted from the pool of items that measure participation.

The final comment from the expert review was on the wording of items CEDL2 and EEGR4. The experts suggested that item CEDL2 should be re-worded to *"I carefully read through the assessment criteria for my programming assessment so that I know what I have to do in order to succeed in my programming course"*. According to the experts, the item wording to *"achieve a particular grade"* suggests that the student may learn enough to obtain a grade they desire, and may not be a deep learning approach.

As for item EEGR4, the experts suggested that the statement *"compared to assessments in other courses"* should be dropped. According to the experts, the statement appears to be double-barreled whereby the statement appears to be measuring the feeling of achievement and also compares with assessments in other courses. Item EEGR4 was therefore re-worded to *"I feel a strong sense of achievement when I complete my programming assessments"*. Thus, the four changes were made to the pool of items, and the next stage of scale development commenced.

6.2.2 Scale Development: Card Sorting

According to Moore and Benbasat (1991, p. 199), the objectives of the scale development stage are to assess the construct validity and to identify items that lack clarity. In the IS discipline, the card sorting method is used for scale development and was initially referred to as Q-sorting (Straub et al., 2004). The card sorting method uses a set of cards containing a statement (to an item) on each card, and the judges group the statements that are similar.

The card sorting method not only assesses the content validity, but also the convergent and discriminant validity of the construct (Straub et al., 2004). In this study, the card sorting method was performed on the engagement constructs and the self-assessment construct. The card sorting method was not applied to the programming self-efficacy construct as the scale had been validated in previous research. The card sorting method performed by Moore and Benbasat (1991) was closely followed in this study.

They performed two rounds of card sorting. In the first round, also referred to as the open round, the judges were given a set of items and were asked to group similar statements by naming the groups of similar items. The purpose of asking several judges to name and define the group of similar statements was to minimise the issue of "interpretational confounding" that was highlighted by (Burt, 1976, p. 4). He explains that the issue of interpretational confounding happens when one assigns a meaning to a latent variable which may differ from the empirical meaning of the latent variable.

In the second round, also referred to as the closed round, a different set of judges (from the 1st round) was asked to match the items to a set of construct names. If in the two rounds, the judges are able to group the items in similar categories, and name them as originally intended, then the confidence in the validity of the constructs increases, and convergent and discriminant validity of the items can be confirmed (Moore & Benbasat, 1991).

6.2.2.1 Round 1: Open Card Sorting

To ensure representativeness from the two countries where data collection efforts were focused, two judges from each country (Malaysia and New Zealand), were invited to perform Round 1 of the card sorting activity. All four judges were academics who had taught or were teaching an introductory programming course. Thus, they were familiar with the activities that a novice programmer undertakes, and the experiences of the novice programmer when learning programming.

Two of the judges were experienced in the card sorting method and did not require any further training on the card sorting activity. The other two judges were briefed on the card sorting method and were referred to a resource on YouTube which explains and visually demonstrates a card sorting activity. The judges were also given a simple trial run of the card sorting activity by using an example of two constructs involving a user's perception of their fitness level and their emotional well-being. Any questions that the judges had were also answered during the training. The training was conducted a week before the actual card sorting activity.

During the actual card sorting activity, each judge was given a set of instructions, and 54 randomly sorted cards containing statements for each scale item. They were then asked to group the statements into the type of construct that the statement or set of statements represented, and asked to give a name that best described the group of statements. The card sorting activity was performed individually, with opportunities to ask questions and seek clarification on the method. No questions were asked during Round 1 of the card sorting activity. The judges took about between 40 and 45 minutes to complete the card sorting activity. Appendix F contains the instructions for the card sorting activity.

Moore and Benbasat (1991) suggested two measurement criteria for examining the outcome of the card sorting activity. Both strategies required the assessment of the inter-rater reliability of the judges. Inter-rater reliability refers to how well the raters agree with each other (Straub et al., 2004). The first measurement criterion uses the Cohen's Kappa test which statistically measures the inter-rater agreement between two raters by removing the chance agreement (Cohen, 1960). A substantial agreement score is ideally > *.7* (Landis & Koch, 1977; Miles & Huberman, 1994). In round one, the average raw agreement scores between the judges were *.903,* while the Cohen's Kappa scores for the inter-rater agreement was *.795* (Table 6.20). The average Cohen's Kappa scores indicate a substantial agreement between the judges.

Table 6.20: Inter-judge agreement scores – Round one

Paired Judges	**Round One**	
	Raw Agreement	**Cohen's Kappa**
1 & 2	.867	.718
1 & 3	.833	.755
1 & 4	.867	.718
2 & 3	.950	.895
2 & 4	.933	.859
3 & 4	.917	.825
Average	**.903**	**.795**

The second measure to assess the reliability and validity of the items was to examine the frequency in which the items were placed into the intended construct (Moore & Benbasat, 1991). The higher the frequency of items placed into the intended construct, the better the inter-rater agreement of the item (Moore & Benbasat, 1991). Table 6.21 presents the items placement score in Round 1 of the card sorting activity. The overall item placement ratio for round one was 91.07%, with most items placed within the intended construct. The intended number of constructs was 11, and only two judges identified all 11 constructs (Table 6.22).

There were differences in the placement of items BEPA1, BEEF3 and CESL4 between the judges and the original intent. Item BEPA1 which was initially a measure for *participation* was grouped as an item to measure *effort* by two judges while item CESL4 which was

initially a measure for *surface learning* was grouped into an item to measure *trial and error* by all four judges. In addition, item BEEF3 which was initially a measure of *effort* was grouped into an item to measure *persistence* by three judges. Upon closer examination of the three items, the items were grouped accurately by the judges. As such, item BEPA1 was moved to the *effort* construct, item BEEF3 was moved to the *persistence* construct, and item CESL4 was moved to the *trial and error* construct.

In addition, all four judges had difficulty placing the item BEPA3 whose intended construct was participation. They were not certain if the statement on discussions outside class time (item BEPA3) could fit into any of the constructs that each judge had created. At this stage, it was obvious that the *participation* construct lacked reliable measures. The findings from the focus groups revealed that students were regularly attending classes, engaged in discussions outside of class time, and contributed to discussion boards. These activities, although interpreted as *participation*, did not appear to concur with the opinion of the judges in the current round of card sorting. Thus, item BEPA3 was removed. With the removal of item BEPA3, there were no more items remaining to measure *participation*. The data from the focus groups were reviewed again in order to ascertain if any new items could be created from the responses of the participants and are suggestive of *participation* in the course. Since no further evidence of participation could be found in the focus group data, the *participation* construct was dropped from being an indicator of behavioural engagement in this study due to the lack of reliable measures.

Next, the naming of the constructs by the judges was examined. All judges named the constructs accurately or close to the intended construct name (Table 6.22). Judge 1 and Judge 3 had placed item BEPA1 into a construct which they suggested may be related to *"taking part in programming related activities"* and being *"proactive"*. These descriptions could be interpreted as being similar to naming the construct as participation. However, no further action was taken since the decision has been made to drop the participation construct due to a lack of agreement on appropriate measures for participation, and BEPA1 had been moved as a measure for effort.

Judge 3 had grouped the items for enjoyment and interest into one construct and explained that the items seemed to indicate that "*programming is fun and interesting"*. Since fun is closely related to enjoyment, and interesting relates to interest, Judge 3 had successfully identified both constructs.

Table 6.21: Items placement score – Round 1

Target Construct	Actual Constructs													
	Part[1]	**HS[2]**	**Eff[3]**	**Pers[4]**	**DL[5]**	**SL[6]**	**TandE[7]**	**Enj[8]**	**Int[9]**	**Grat[10]**	**Self-asst[11]**	**N/A***	**Total**	**Target %**
Participation[1]	2		2									4	8	25
Help-seeking[2]		20											20	100
Effort[3]			21	3									24	87.5
Persistence[4]				20									20	100
Deep Learning[5]					16								16	100
Surface Learning[6]					1	27	4						32	84.37
Trial and Error[7]						1	19						20	95
Enjoyment[8]								16					16	100
Interest[9]								5	15				20	75
Gratification[10]										20			20	100
Self-assessment[11]											20		20	100
Total Item Placements: 216					Hits: 196 (54 cards, 4 judges)						Overall Hit Ratio: 91.07%			

**N/A refers to items that could not be grouped or ambiguous*

Table 6.22: Construct names provided by judges – Round 1

Initial Constructs	Judge			
	1	**2**	**3**	**4**
Participation	Take part in programming related activities		Proactive approach to learning programming	
Effort	Putting in the time to work on the assessment	Practise programming	Make an attempt to learn programming	Set aside time to learn programming
Help-seeking	Reach out for help	Ask for help when stuck in programming	Ask for help	Help
Persistence	Do not give up learning programming	Determination to succeed	Keep trying to learn programming	Persist
Deep Learning	Strategies to learn	Good strategy to learn programming	Effective problem-solving	Deep thinking
Surface Learning	Learning using memory. Weak understanding	Strategy to learn	Less effective learning strategy	Rote learning
Trial and Error	Technique to solve programming errors	Practise programming	Step-by-step analysis of code	Try various strategies
Enjoyment	Enjoy	Enjoy programming	Programming is fun and interesting	Enjoy
Interest	Passion	Deep interest		Keen on programming
Gratification	Excitement	Euphoria	An overwhelming sense of happiness when achieving something	Elated/Feeling extremely pleased
Self-assessment	My performance	Time and effort put into learning programming	Perception about assessment	How well I performed

6.2.2.2 Round 2: Closed Card Sorting

In Round 2 of the card sorting activity, three judges (different from Round 1) were invited to participate in the card sorting activity. One judge was an academic who had experience teaching introductory programming courses while the other two judges were Postgraduate students with programming experience. Students were selected to perform the card sorting in order to ensure that the novice programmers (the actual participants in this study) could understand and relate to the activities and experiences described in the statements. All three judges did not have any experience in the card sorting method. Thus, a similar training in Round 1 was given to the judges.

During the actual card sorting activity, each judge was given a set of instructions, 53 randomly sorted cards containing statements for each scale item, and 10 cards containing the names of each construct and the definition. The judges were asked to match the statement to the construct name and the definition. The card sorting activity was performed individually, with opportunities to ask questions and seek clarification on the method. No questions were asked during Round 2 of the card sorting activity. The judges took about 25 minutes to complete the card sorting activity. Appendix F contains the instructions for the card sorting activity.

The same measurement criteria from Round 1 were used in Round 2. The average raw agreement scores between the judges were *.967* while the Cohen's Kappa scores for the inter-rater agreement was *.875* (Table 6.23). The average Cohen's Kappa scores indicate a substantial agreement between the judges and had improved from Round 1.

Table 6.23: Inter-judge agreement scores – Round 2

Paired Judges	**Round Two**	
	Raw Agreement	**Cohen's Kappa**
1 & 2	.967	.870
1 & 3	.950	.813
2 & 3	.983	.942
Average	**.967**	**.875**

The second measure examined the frequency in which the items were placed into the intended construct. Table 6.24 presents the items placement score in Round 2 of the card sorting activity. The overall item placement ratio for round two was 98.18%, with most items placed within the intended construct. In total, the intended number of constructs was 10. Although all four judges identified 10 constructs, two judges had placed the item CESL6 into the *trial and error* construct while one judge placed item CESL7 into the *deep learning* construct. Upon consultation with the judges, they agreed that items CESL6 and CESL7 were better measures of the *surface learning* construct. Thus, after Round 1 and Round 2 of the card sorting activity, all items had been assessed for content validity, convergent validity, and discriminant validity.

Table 6.24: Items placement score – Round 2

Target Construct	Actual Constructs												
	HS[1]	**Eff[2]**	**Pers[3]**	**DL[4]**	**SL[5]**	**TandE[6]**	**Enj[7]**	**Int[8]**	**Grat[9]**	**Self-asst[10]**	**N/A**	**Total**	**Target %**
Help-seeking[1]	15											15	100
Effort[2]		18										18	100
Persistence[3]			18									18	100
Deep Learning[4]				12								12	100
Surface Learning[5]				1	18	2						21	85.7
Trial and Error[6]						18						18	100
Enjoyment[7]							12					12	100
Interest[8]								15				15	100
Gratification[9]									15			15	100
Self-assessment[10]										15		15	100
Total Item Placements: 159				Hits: 156 (53 cards, 3 judges)							Overall Hit Ratio: 98.18%		

**N/A refers to items that could not be grouped or ambiguous*

6.3 Testing the Questionnaire

The Phase 1 and Phase 3 questionnaires were tested for content and construct validity in two stages. The first stage involved a small number of participants to pre-test the questionnaires in order to ensure that the questionnaire had been appropriately designed for the intended participants (Moore & Benbasat, 1991). The second stage involved a pilot study to ensure that the scales used in the questionnaire were reliable (Moore & Benbasat, 1991), and if the questionnaires worked as per its original intent (Zikmund et al., 2010). When testing the questionnaires, selecting an appropriate sample and the sample size were two factors that were considered (MacKenzie et al., 2011), and are discussed in the following sub-sections.

6.3.1 Pre-test of the questionnaires

A convenient sample of ten undergraduate students who did not have any programming experience was selected to pre-test the Phase 1 questionnaire. Since the Phase 1 questionnaire is administered at the beginning of the introductory programming course, the students were asked to assume that they had recently enrolled in an introductory programming course, and to comment on the design and clarity of the questionnaire. The questionnaire was administered online using Qualtrics software (Qualtrics, Provo, UT), and the link to the questionnaire was e-mailed to the students. The average time that the students took to complete the questionnaire was 8 minutes, which was within the expected completion time of 10 minutes. The feedback from the students was generally positive, with two students picking up some inconsistencies in the formatting and a spelling error. The positive comments include "*nice, short and clear questions*", *"a lot better questionnaire than most questionnaires"*, *"flow is understandable and logical"*, *"Good questionnaire for students who are new to programming. The questions are easy to follow and in good order. I was easily able to understand what each of the questions asked"*. Thus, once the formatting inconsistencies and the spelling error were rectified, no further changes were made to the Phase 1 questionnaire.

Like the Phase 1 questionnaire, a convenient sample of five undergraduate students was chosen to pre-test the Phase 3 questionnaire, but who had taken an introductory programming course. The students were asked to comment on the design and clarity of the questionnaire. The questionnaire was administered online using Qualtrics software (Qualtrics, Provo, UT), and the link to the questionnaire was e-mailed to the students. The average time that the students took to complete the questionnaire was 16 minutes, which was within the expected completion time of 20 minutes. The feedback from the students was generally positive, with one student picking up a question that had been repeated, and inconsistencies in the formatting. The positive comments include *"questions are easy to follow and in good order", "I was easily able to understand what each of the questions asked"*. Thus, once the formatting inconsistencies were rectified, and the duplicate question was removed, no further changes were made to the Phase 3 questionnaire.

6.3.2 Pilot Study

A pilot study is a small scale version of the actual research and is aimed at testing the research instrument while increasing the possibility of a successful outcome (van Teijlingan & Hundley, 2001; Zikmund et al., 2010). According to Zikmund et al. (2010), a pilot study can help improve the quality of the instrument, minimise risks during an actual study should the research instrument be flawed, and confirm the face validity of the instrument items. Face validity refers to "the subjective agreement among professionals that a scale logically reflects the concept being measured" (Zikmund et al., 2010, p. 307), and is normally established before the theory is tested using a confirmatory factor analysis (CFA) (Hair et al., 2010, p. 688).

Prior to conducting the pilot study, the sample size and the administration of the pilot test were considered. Next, the type of tests to run on the pilot test data was considered. Moore and Benbasat (1991) suggested examining the reliability of the data while other researchers suggested that an exploratory factor analysis (EFA) should be performed to examine the factor loadings of the items (MacKenzie et al., 2011).

6.3.2.1 Sample size

The general consensus for the sample size to conduct a pilot study should be large in order to minimise the variance in the responses (DeVellis, 2012). Nunally (1978) suggests 300 as a large sample size. By contrast, DeVellis (2012, p. 102) argues that the number of items and the number of scales should be considered when deciding on the sample size while Hair et al. (2010) recommend using item-to-response ratios that range between 1:5 and 1:10 for exploratory factor analysis (EFA), and the sample should contain no fewer than 50 responses. On the other hand, Comrey and Lee (1992) recommend a sample size between 100 and 500 for an EFA while MacCallum, Widaman, Zhang and Hong (1999) argue that a small sample size of 60 to 100 is adequate if the communalities are high and the factor loadings are strong.

Although the pilot study attempted to achieve a sample size within the ranges recommended in the literature, this proved to be difficult as there were only two cohorts of introductory programming courses available to test the questionnaires. In addition, the response rates dropped during the Phase 3 survey questionnaire resulting in only 55 responses in the final set of usable responses. Nevertheless, the sample size of 55 meets the minimum requirement sample size of 50 that was suggested by Hair et al. (2010).

6.3.2.2 Administration of pilot test

A pilot test should use a sample that is similar to the actual target population of the research (Moore & Benbasat, 1991; Zikmund et al., 2010). Since the target population of this study was students enrolled in an introductory programming course in Malaysia or in New Zealand, the pilot test used a sample of students that fitted the same profile in both countries.

The Phase 1 and Phase 3 questionnaires were piloted between July and October 2013 at the Victoria University of Wellington in New Zealand, and at the Asia Pacific University of Technology and Innovation in Malaysia. Students who were enrolled in an introductory programming course at both HEIs were invited to participate in the research. The survey was administered online using Qualtrics (Qualtrics, Provo, UT).

Upon receiving approval from the Human Ethics Committee (Appendix A), the Phase 1 of the survey questionnaire was administered during Week 3 of the introductory programming course. During the lecture, the researcher was given 15 minutes to brief the students, and to invite the students to participate in the research. The Participant Consent Form and the Participant Information Sheet were distributed. The participants were requested to complete the Participant Consent Form, and to return the form to the researcher. Since the researcher could not be present in Malaysia to brief the students, the course instructor acted as a proxy to the researcher, who then briefed, and administered the questionnaire to the participants. To encourage the students to participate in the survey, the participants were entered into a draw which offered monetary rewards to the lucky winners. The link to the questionnaire was e-mailed to the participants once the students completed and returned the Participant Consent Form to the researcher, and the link to the questionnaire remained open for one week. A reminder was sent to the participants to complete the questionnaire four days after the link was opened. Ninety-eight (98) students participated in the Phase 1 survey questionnaire.

In the final week of the course (Week 12), the students that had participated in the pilot study of the Phase 1 questionnaire were invited to participate in the Phase 3 survey questionnaire. An e-mail was sent to the students which contained a link to the Phase 3 questionnaire. The questionnaire was administered online using Qualtrics. The link remained open for two weeks, with a reminder sent to the participants a week after the first invitation to complete the questionnaire. Fifty-eight (58) students participated in Phase 3 of the survey. But, only 55 usable sets of responses (completed Phase 1 and Phase 3 survey) could be used to analyse the reliability of the scales.

6.3.2.3 Descriptive Statistics

Table 6.25 presents the response rate by country. Of the 163 students that were approached to participate in the pilot study at the beginning of the course, 55 usable sets of responses (completed Phase 1 & Phase 3 questionnaire) were eventually used. The number of participants from Malaysia and New Zealand was almost equal, with 57.1% of the participants from HEIs in New Zealand.

In Phase 1 of the data collection, the overall response rate was 60.1% with a significantly lower response rate in New Zealand (46.7%) compared to Malaysia (97.7%). By Phase 3, 40.8% of the participants had dropped out, with 41.1% of the participants in New Zealand dropping out in Phase 3 while 40.5% of the participants had dropped out in Malaysia.

Table 6.25: Response Rate by Country – pilot test

Country	Sample Population	No. of Participants in Phase 1	No. of Participants in Phase 3	No. of Participants completed Phase 1 & 3
New Zealand	120 (73.6%)	56 (57.1%)	33 (56.9%)	31 (56.4%)
Malaysia	43 (26.4%)	42 (42.9%)	25 (43.1%)	24(43.6%)
Total	**163**	**98**	**58**	**55**

Table 6.26: Descriptive Statistics of Participants – pilot test

Gender		
	Frequency (n = 55)	**Percentage (%)**
Male	36	65.5
Female	19	34.5
Age		
25 years or less	54	98.2
More than 25 years	1	1.8
Ethnicity		
New Zealand European	18	32.7
Other European	-	-
New Zealander	1	1.8
Maori	1	1.8
Pasifika	1	1.8
Asian	31	56.4
Middle Eastern	-	-
Latin American	-	-
African	-	-
Others	3	5.5
Programming Experience		
Yes	15	27.3
No	40	72.7
Type of Major		
Major	46	83.6
Non-Major	9	16.4

Table 6.26 presents the descriptive statistics of the participants. A large proportion of the participants were males (65.5%). 98.2% of the participants were less than 25 years old. Half (56.4%) of the participants identified themselves as Asians. 72.7% of the participants did not have prior programming experience, and the majority (83.6%) of the participants were from the Computer Science related majors.

6.3.2.4 Test for Normality

Before proceeding with examining the reliability of the scales, the data was tested for normality. Although the PLS-SEM data analysis technique that was used to analyse the data in this study is able to handle non-normally distributed data, the data was still tested for normality to check that there were no extreme occurrences of non-normal data (Chin 1998a; Hair et al., 2014). Skewness and kurtosis are two values that are analysed to determine if the distribution of the data is normal (Field, 2013). Skewness "assesses the extent to which a variable's distribution is symmetrical" (Hair et al., 2014, p. 54) while kurtosis "is a measure of whether the distribution is too peaked" (Hair et al., 2014, p. 54). To determine the normality of the distribution, the skewness and kurtosis values should be close to zero (0) (Field, 2013; Hair et al., 2014), and may be within an acceptable range of +/- 2 (Gravetter & Wallnau, 2014). Table 6.27 displays the skewness and kurtosis values of each construct in the research model. The skewness of the constructs was between -1.084 and .611 while the kurtosis of the constructs was between -1.246 and .740. These values are within the acceptable range of +/- 2, confirming that the data from the pilot test was normally distributed.

Table 6.27: Skewness and Kurtosis - Distribution of Data

Higher-Order Construct	Lower-order Construct	Skewness		Kurtosis	
		Statistic	Std. Error	Statistic	Std. Error
Pre-Programming Self-Efficacy	General self-efficacy	.156	.322	-.592	.634
	Independence and persistence				
	Complex programming tasks				
	Self-regulation				
	Simple programming tasks				
Post-Programming Self-Efficacy	General self-efficacy	-.096		-1.246	
	Independence and persistence				
	Complex programming tasks				
	Self-regulation				
	Simple programming tasks				
Behavioural Engagement	Help Seeking	-.119		-.072	
	Effort	-.170		-.856	
	Persistence	.026		-.601	
Cognitive Engagement	Deep Learning	-.350		-.544	
	Surface Learning	.313		.035	
	Trial and Error	.611		-.657	

Higher-Order Construct	Lower-order Construct	Skewness		Kurtosis	
		Statistic	Std. Error	Statistic	Std. Error
Emotional Engagement	**Interest**	-.080		-.534	
	Gratification	-.409		-.868	
	Enjoyment	-1.084		.740	
	Self-Assessment	-.089		-.935	
	Programming Grade	-.975		-.409	

6.3.2.5 Reliability Analysis

Straub et al. (2004) argue that measurement instruments must be assessed for their reliability and validity as it is important to ensure that the latent variables are represented accurately. A reliable instrument performs consistently and in a predictable manner when repeated (DeVellis, 2012; Field, 2013; Hair et al., 2010). While validity examines the measurements between constructs, reliability, on the other hand, examines the measurements within a construct (Straub et al., 2004, p. 399).

The internal consistency reliability is a commonly used reliability measure (Hair et al., 2010), and was used to assess the reliability of the constructs in this study. Other types of reliability measures such as using split halves, test-retest, alternative or equivalent forms, and unidimensional were not assessed in this study since these measures were either more traditional than the internal consistency reliability measure, or better suited for qualitative research, or were relatively new and optional methods to assess the reliability of a construct (Straub et al., 2004).

Hair et al. (2010, p. 125) suggested three ways to measure internal consistency. The first measure of internal consistency examines the correlation of the items to the summated scale score (item-total correlation) and the correlation between the items (inter-item correlations). An acceptable item-total correlation should be > .5, and the inter-item correlation should be > .3 (MacCallum, Roznowski, Mar, & Reith, 1994).

The second measure of internal consistency uses the Cronbach's alpha estimate (Cronbach, 1971). An acceptable Cronbach's alpha estimate should be > .7 (Kline, 1999; Nunnally & Bernstein, 1994). However, a Cronbach's alpha estimate of .6 is also acceptable in psychological constructs due to its diversity and exploratory nature (Kline, 1999; Nunally, 1978; Nunnally & Bernstein, 1994) and generally indicates fair reliability (Zikmund et al., 2010). In addition, a Cronbach's alpha estimate that is between .5 and .6 may be acceptable in exploratory research or at an early stage of research (Nunnally, 1978).

The next measure of internal consistency is the composite reliability (CR) and average variance extracted (AVE). These are typically performed during confirmatory factor analysis (CFA). This measure is discussed in Chapter 7, Section 7.5.2.

To examine the reliability of the constructs in the pilot study, the Cronbach's alpha, item-total correlations, and the inter-item correlations were examined. According to Moore and Benbasat (1991), in order to improve the Cronbach's alpha estimate, an item with the lowest item-total correlation, and inter-item correlation may be considered as a candidate for deletion. Thus, the following rules were applied when the constructs were assessed for their reliabilities:

1. If initial Cronbach's alpha is > .6 and does not improve when items are deleted, then accept items.
2. If the Cronbach's alpha is < .6, then
    a. Delete item with the lowest item-total correlation. Field (2013, p. 715) suggests that item-total correlations that are < .3 should be deleted as the item does not contribute positively to the reliability of the construct.
3. Inter-item correlations > .3.

Table 6.28 presents the reliability of the indicators of the higher-order pre-programming self-efficacy construct while Table 6.29 presents the reliability of the indicators of the higher-order post-programming self-efficacy construct. The Cronbach's alpha estimate for the lower-order constructs *general self-efficacy, independence and persistence*, *complex programming tasks*, *self-regulation*, and *simple programming tasks* were well above .*7*, thereby confirming that the reliability of the items in the pre- and post-programming self-efficacy constructs were excellent.

The item-total correlations for the lower-order pre- and post-programming self-efficacy constructs were > .5 with the exception of items BPSE12, BPSE13 and APSE12. Since the item-total correlation for APSE12 (.489) was only marginally lesser than .5, the item was retained for further analysis. The items BPSE12 and BPSE13 were also retained for two reasons. First, the deletion of both the items did not result in a significantly higher Cronbach's alpha estimate, and second, the same items demonstrated stronger item-total correlations (measured by APSE12 and APSE13) in the post-programming self-efficacy construct (Table 6.29). Thus, it was felt that, at this stage, items BPSE12 and BPSE13 may yield better item-total correlations when data is collected from a larger sample of the population.

The inter-item correlations for all items were > .3 with the exception of items BPSE12 and BPSE13. Here again, no further decision was made to delete the items since the same items demonstrated stronger inter-item correlations (measured by APSE12 and APSE13) in the post-programming self-efficacy construct.

Table 6.28: Reliability analysis of the pre-programming self-efficacy lower-order constructs

Lower-order construct	Higher-order construct: Pre-Programming Self-Efficacy Item	Item-Total Correlation	Cronbach's Alpha if Item Deleted
General self-efficacy	**Cronbach α = .894**		
	BPSE1	.734	.873
	BPSE2	.756	.870
	BPSE3	.667	.884
	BPSE4	.720	.876
	BPSE5	.727	.874
	BPSE6	.712	.877
Independence and persistence	**Cronbach α = .826**		
	BPSE7	.617	.796
	BPSE8	.716	.777
	BPSE9	.768	.767
	BPSE10	.642	.790
	BPSE11	.716	.779
	BPSE12	*.440*	*.823*
	BPSE13	*.163*	*.866*
Complex programming tasks	**Cronbach α = .832**		
	BPSE14	.666	.788
	BPSE15	.696	.781
	BPSE16	.502	.832
	BPSE17	.738	.767
	BPSE18	.565	.818
Self-regulation	**Cronbach α = .769**		
	BPSE19	.633	-
	BPSE20	.633	-
Simple programming tasks	**Cronbach α = .894**		
	BPSE21	.747	.874
	BPSE22	.791	.860
	BPSE23	.749	.869
	BPSE24	.730	.874
	BPSE25	.711	.878

Table 6.29: Reliability analysis of the post-programming self-efficacy lower-order constructs

Lower-order construct	Higher-order construct: Post-Programming Self-Efficacy		
	Item	**Item-Total Correlation**	**Cronbach's Alpha if Item Deleted**
General self-efficacy	**Cronbach α = .879**		
	APSE1	.722	.852
	APSE2	.683	.859
	APSE3	.667	.861
	APSE4	.569	.877
	APSE5	.713	.854
	APSE6	.770	.843
Independence and persistence	**Cronbach α = .857**		
	APSE7	.664	.831
	APSE8	.684	.828
	APSE9	.742	.821
	APSE10	.515	.851
	APSE11	.666	.830
	APSE12	.489	.854
	APSE13	.604	.840
Complex programming tasks	**Cronbach α = .875**		
	APSE14	.752	.840
	APSE15	.695	.850
	APSE16	.627	.866
	APSE17	.744	.838
	APSE18	.717	.845
Self-regulation	**Cronbach α = .840**		
	APSE19	.736	-
	APSE20	.736	-
Simple programming tasks	**Cronbach α = .870**		
	APSE21	.786	.820
	APSE22	.775	.822
	APSE23	.588	.867
	APSE24	.627	.859
	APSE25	.705	.841

Table 6.30 presents the reliability of the lower-order behavioural engagement constructs. The Cronbach's alpha value for *help-seeking* and *effort* were < .5, suggesting weak reliability. The lowest item-total correlation in these constructs appeared to be the negatively worded items which had been reverse-coded. The Cronbach's alpha estimate improved when the negatively worded items in these constructs were deleted.

The initial Cronbach's alpha value for *help-seeking* improved from .446 to .686 when the reverse-coded negatively worded items BEHS2* and BEHS4* were deleted while the initial Cronbach's alpha value for *effort* improved from .371 to .623 when the reverse-coded negatively worded item BEEF4*, and item BEEF1 was deleted. The Cronbach's alpha did not improve when more items were deleted. Thus, the remaining items BEEF2, BEEF5, BEEF6, and BEPA1 were retained. For *persistence*, the initial Cronbach's alpha value improved from .703 to .718 when the reverse-coded negatively worded item BEPE2* was deleted. Overall, the item-total correlations for the remaining items in *help-seeking*, *effort* and *persistence* improved to > .5, and the inter-item correlations for all items were > .3 after deletion of the items with weak item-total correlations.

Table 6.30: Reliability analysis of the indicators of behavioural engagement

Higher-order construct: Behavioural Engagement			
Lower-order construct	**Item**	**Item-Total Correlation**	**Cronbach's Alpha if Item Deleted**
Help-seeking	**Cronbach α = .446/.686 (BEHS2* & BEHS4* deleted)**		
	BEHS1	.369	.282
	*BEHS2**	*.011*	*.541*
	BEHS3	.319	.333
	*BEHS4**	*.281*	*.354*
	BEHS5	.220	.401
Effort	**Cronbach α = .371/.623 (BEEF4* & BEEF1 deleted)**		
	BEEF1	*-.005*	*.453*
	BEEF2	.512	.137
	*BEEF4**	*-.190*	*.554*
	BEEF5	.248	.276
	BEEF6	.508	.056
	BEPA1	.231	.328
Persistence	**Cronbach α = .703/.718 (BEPE2* deleted)**		
	BEPE1	.404	.673
	*BEPE2**	*.287*	*.718*
	BEPE3	.488	.646
	BEPE4	.469	.654
	BEPE5	.573	.618
	BEEF3	.427	.666

*Items marked with * are negatively worded items*

Table 6.31 presents the reliability of the lower-order cognitive engagement constructs. The Cronbach's alpha estimate for *deep learning* and *trial and error* were > .7, suggesting excellent reliability. None of the items that measure *deep learning* and *trial and error* were

deleted as deletion of any of the items would not result in a higher Cronbach's alpha estimate. However, the item-total correlation for item CESL1 and CESL7 was < .3. When these two items were deleted, the Cronbach's alpha estimate improved from .638 to .732.

Table 6.31: Reliability analysis of the indicators of cognitive engagement

Higher-order construct: Cognitive Engagement			
Lower-order construct	**Item**	**Item-Total Correlation**	**Cronbach's Alpha if Item Deleted**
Deep Learning	**Cronbach α = .712**		
	CEDL1	.481	.661
	CEDL2	.424	.694
	CEDL3	.584	.599
	CEDL4	.513	.642
Surface Learning	**Cronbach α = .638/.732 (CESL1 & CESL7 deleted)**		
	CESL1	*.187*	*.653*
	CESL2	.522	.547
	CESL3	.451	.567
	CESL5	.487	.552
	CESL6	.422	.581
	CESL7	*-.079*	*.704*
	CESL8	.465	.563
Trial and Error	**Cronbach α = .710**		
	CETE1	.354	.702
	CETE2	.488	.656
	CETE3	.367	.694
	CETE4	.551	.642
	CETE5	.447	.670
	CESL4	.477	.661

*Items marked with * are negatively worded items*

Table 6.32 presents the reliability of the lower-order emotional engagement constructs. The Cronbach's alpha estimate of .837 was excellent for *gratification*. Therefore, all items that measures *gratification* were retained. The initial Cronbach's alpha for *enjoyment* improved from .539 to .623 when the reverse-coded negatively worded item EEEN4* was deleted.

The Cronbach's alpha did not improve when the remaining items were considered for deletion. Therefore, EEEN1, EEEN2, and EEEN3 were retained in the lower-order enjoyment construct. The inter-item correlations for all items were > .3. The initial Cronbach's alpha value for *interest* improved from .352 to .715 when the reverse-coded negatively worded item EEIN5* and item EEIN2 were deleted. The deletion of the two items

improved the item-total correlations of the remaining items to > .5. The inter-item correlations for all items were > .3.

Table 6.32: Reliability analysis of the indicators of emotional engagement

Higher-order construct: Emotional Engagement			
Lower-order construct	**Item**	**Item-Total Correlation**	**Cronbach's Alpha if Item Deleted**
Enjoyment	**Cronbach α = .539/.623 (EEEN4* deleted)**		
	EEEN1	.482	.307
	EEEN2	.555	.205
	EEEN3	.220	.543
	*EEEN4**	*.097*	*.623*
Gratification	**Cronbach α = .837**		
	EEGR1	.655	.801
	EEGR2	.633	.806
	EEGR3	.656	.801
	EEGR4	.452	.854
	EEGR5	.834	.742
Interest	**Cronbach α = .352/.715 (EEIN5*& EEIN2 deleted)**		
	EEIN1	.253	.225
	EEIN2	*-.044*	*.492*
	EEIN3	.333	.191
	EEIN4	.570	-.019
	*EEIN5**	*-.059*	*.487*

*Items marked with * are negatively worded items*

Table 6.33 presents the reliability of the *self-assessment* construct. The initial Cronbach's alpha value improved from .719 to .849 when the reverse-coded negatively worded items SA3* and SA4* were deleted. The programming grade construct consists of only one item. Thus, the Cronbach's alpha estimate was not applied to this construct.

Table 6.33: Reliability analysis of self-assessment and programming grade

Programming Performance			
Constructs	**Item**	**Item-Total Correlation**	**Cronbach's Alpha if Item Deleted**
Self-assessment	**Cronbach α = .719/.849 (SA3* & SA4* deleted)**		
	SA1	.583	.629
	SA2	.646	.604
	*SA3**	*.488*	*.706*
	*SA4**	*.380*	*.668*
	SA5	.384	.737
Programming Grade	**Cronbach α = .687**		
	ProgGrade *(one item indicator)*	-	-

*Items marked with * are negatively worded items*

Overall, the item-total correlation of the reverse-coded negatively worded items was weak during the reliability analysis of the pilot test data. The Cronbach's alpha estimates improved significantly when the reverse-coded negatively worded items were deleted from the construct. The negatively worded items were examined again to ensure that the values were accurately reverse-coded for analysis. No plausible explanation could be found for the poor performing negatively worded items. The literature on questionnaire design was then re-examined for a possible explanation of the weak correlations. DeVellis (2012) observed that negatively worded items tended to perform poorly and that "the disadvantages of negatively worded items outweigh the benefits" as "reversals in item polarity tend to confuse the respondents particularly in a long questionnaire" (p. 84). This explains the poor performing negatively worded items in this study and justifies the exclusion of these items from the questionnaire.

6.3.2.6 Factor Analysis

Factor analysis is a multivariate data analysis technique that uses statistical methods to reduce a set of correlated items into a number of variables called factors (Hair, Black, Babin, & Anderson, 2010; Zikmund et al., 2010), and is used to test construct validity (Straub et al., 2004). Although the main purpose of a factor analysis is to identify the number of latent variables that determine a set of items, a factor analysis also serves other purposes such as to reduce the number of variables, determine the meaning of the factors, and to examine the performance of the items (DeVellis, 2012; Field, 2013; Hair et al., 2010).

There are two types of factor analysis – exploratory factor analysis (EFA) and confirmatory factor analysis (CFA). The purpose of an EFA is to determine the number of factors in a set of variables while the CFA is performed to test the construct validity of a model by examining how the theoretical structure of the factors fit into the actual observed structure of the factors (Zikmund et al., 2010, p. 593). This would require an examination of the factor loadings of the indicators in order to determine the best set of indicators that reflect the latent variables (Schreiber, Nora, Stage, Barlow, & King, 2006). An EFA may be used in a subset of the actual sample in order to examine the structure of the items and to set up the measurement model (Hair et al., 2010, p. 680). The CFA on the other hand is used to test the measurement model and validate the measurement theory (Hair et al., 2010, p. 680).

In IS research, convergent and discriminant validity is commonly assessed by performing an EFA while convergent and discriminant validity in SEM-PLS is assessed by performing a CFA (DeVellis, 2012; Schreiber et al., 2006; Straub et al., 2004). In addition, an EFA would be appropriate for measurement scales which have not been pre-validated (Bagozzi & Philips, 1982) while Straub et al. (2004) stressed that at least one (or more) techniques should be used to assess the convergent and discriminant validity of the constructs. Since the research model in this study contains pre-validated and new measurement scales, the measurement model was assessed using both the EFA and the CFA in the final analysis of the data. However, for the pilot test, the EFA was performed to examine the construct validity in this study.

6.3.2.7 Exploratory Factor Analysis (EFA)

The first consideration in performing an EFA is to determine the number of factors to extract. The number of factors to extract can be determined from the eigenvalues, or based on the theoretically-driven number of constructs specified in the research model, or based on the argument that the percentage of variance explained should be more than 60% (Hair et al., 2010; Zikmund et al., 2010). The second option, to use a theoretically-driven number of factors to extract, was selected as the main criterion for extraction for the EFA since the number of factors had been determined in the revised research model (Chapter 5, Section 5.4). However, the eigenvalues and the percentage of variance were also examined in order to confirm the appropriateness of the main criteria that was selected.

Next, the type of extraction method was considered. There are two types of extraction methods: the *common factors analysis* or the *component analysis* (Hair et al., 2010). When deciding on which extraction method to use, the following two factors must be considered: a) the objective of the factor analysis; and b) how much is previously known about the variance in the variables (Hair et al., 2010, p. 107).

A component analysis extraction method is appropriate when the purpose of the analysis is to "summarize most of the original information (variance) in a minimum number of factors for prediction purposes" (Hair et al., 2010, p. 107) while a common factor analysis method is appropriate when the purpose of the analysis is to "identify underlying factors or dimensions that reflect what the variables share in common" (Hair et al., 2010, p. 107). The *principal component analysis (PCA)* extraction method was used to perform the EFA because the main purpose of the pilot test is to determine the minimum number of factors required to maximise the variance in the variables (Hair et al., 2010, p. 107), and is the most popular extraction method used in IS research (Straub et al., 2004).

The third consideration was the type of rotation method. There are two types of rotation methods: the *orthogonal rotation methods* or the *oblique rotation methods* (Hair et al., 2010). The orthogonal rotation methods are commonly used when the purpose of the analysis is to reduce the data or to derive measures that are not correlated for subsequent multivariate analysis (Hair et al., 2010, p. 116). On the other hand, the oblique rotation methods are commonly used when the purpose of the analysis is to derive meaningful factors or constructs (Hair et al., 2010, p. 116). Field (2013) suggests that the oblique rotation methods are appropriate for items that measure psychological constructs since psychological constructs tend to correlate between factors. Since the main purpose of the pilot study was to derive uncorrelated measures for subsequent analysis, the *varimax* rotation, which is an orthogonal rotation method, was selected as the rotation method in the pilot study.

Hair et al. (2010) argued that the factor loadings of the EFA is "conservative and should only be used as a starting point to include a variable for further consideration" (p. 118). Thus, the following criteria for the assessment of the factors were used in the EFA of the pilot data:

a. The Kaiser-Meyer-Olkin (KMO) measure of sampling adequacy tests the "ratio of squared correlation between variables to the squared partial correlation between variables" (Field, 2013, p. 684). The acceptable threshold for the KMO measure should be > .5 (Field, 2013).
b. The Significance of Bartlett's test of Sphericity tests if the correlation between variables is significantly different. The acceptable threshold for the Bartlett's test should be < .05 (Field, 2013).
c. The threshold for factor loadings is dependent on the sample size. In a sample size of 60, the factor loading should ideally be > .7 for significance at the .05 level while an item with a factor loading of > .5 is acceptable in a sample size of 100 or more (Hair et al., 2010. p. 117).

Although the criteria above were used to assess the factors in the EFA, Hair et al. (2010) suggested that researchers should also ensure that the communalities of the items are > .5, and perform different rotation methods when cross-loadings occur.

Subsequently, in the interpretation of the factor model, researchers can choose not to delete an item that did not meet the criteria above if the purpose of the factor analysis is for data reduction (Hair et al., 2010, p. 120). The EFA was performed between the programming self-efficacy construct and the lower-order engagement constructs, and between the lower-order engagement constructs and the programming performance construct since Straub et al. (2004) suggested that the EFA be performed on each directional stage of the research model. Although the programming self-efficacy scale had been validated previously, an EFA was nevertheless performed to confirm the factors in the programming self-efficacy scale.

Programming self-efficacy

The factor loadings of the 25 items in the pre-programming self-efficacy scale were examined and are presented in Table 6.34. A varimax rotation on the pre-programming self-efficacy items resulted in a KMO score that was > .5 (KMO = .797) and the significance of the Bartlett's test that was < .05 (p = .000). The rotation produced 4 factors. All items in the *general self-efficacy* construct loaded strongly onto the intended construct (BPSE1-BPSE6). Items BPSE7 – BPSE11 loaded strongly on the *independence and persistence* construct. However, items BPSE12 and BPSE13 loaded strongly on the *general self-efficacy* construct and *complex programming tasks* construct (BPSE13 only) instead of the *independence and persistence* construct. Interestingly, the items BPSE21 - BPSE25 loaded strongly on the *complex programming tasks* construct instead of the *simple programming tasks* construct. Since the items were validated in a previous research, all items were retained in their intended construct and the factor loadings will be re-examined during the CFA. The total variance explained by the 4 factors was 69.84%. Item loadings were above the threshold of .5.

Table 6.34: Factor loadings for pre-programming self-efficacy – pilot test

Items	Factor				
	General self-efficacy	**Independence and persistence**	**Complex programming tasks**	**Self-regulation**	**Simple programming tasks**
BPSE1	.737				
BPSE2	.722				
BPSE3	.678				
BPSE4	.791				
BPSE5	.672				
BPSE6	.727				
BPSE7		.725			
BPSE8		.885			
BPSE9		.876			
BPSE10		.802			
BPSE11		.864			

Items	Factor				
	General self-efficacy	**Independence and persistence**	**Complex programming tasks**	**Self-regulation**	**Simple programming tasks**
BPSE12	*.673*				
BPSE13	*.641*		*.557*		
BPSE14			.659		
BPSE15			.596		
BPSE16			.653		
BPSE17			*.777*		
BPSE18			.725		
BPSE19				.789	
BPSE20				.851	
BPSE21			*.647*		
BPSE22			*.702*		
BPSE23			*.660*		
BPSE24			*.631*		
BPSE25			*.754*		

Kaiser-Meyer-Olkin Measure of Sampling Adequacy = .797 Bartlett's test of Sphericity Approx. Chi-square = 991.305 / df = 300 / Sig. = .000

The factor loadings of the 25 items in the post-programming self-efficacy scale were examined and are presented in Table 6.35. A varimax rotation on the pre-programming self-efficacy items resulted in a KMO score that was > .5 (KMO = .827) and the significance of the Bartlett's test was < .05 (p = .000). Like the pre-programming self-efficacy construct, the rotation produced 4 factors. All items in the *general self-efficacy* construct loaded strongly onto the intended construct, with the exception of APSE4 which loaded strongly on the *complex programming tasks* construct. Items APSE7 – APSE11 loaded strongly on the *independence and persistence* construct. However, items APSE12 and APSE13 loaded strongly on the *complex programming tasks* construct instead of the *independence and persistence* construct. Item APSE17 loaded strongly on the *self-regulation* construct, and with a slightly lower factor loading on the *complex programming tasks* construct. Interestingly, the items APSE21 - APSE25 loaded strongly on the *complex programming tasks* construct instead of the *simple programming tasks* construct. Like the pre-programming self-efficacy construct, the items were validated in a previous research, and were retained in their intended construct and the factor loadings will be re-examined during the CFA. The total variance explained by the 4 factors was 67.47%. Item loadings were above the threshold of .5 with the exception of APSE23 and APSE24 which were slightly below the .5 threshold (.457 and .494 respectively).

Table 6.35: Factor loadings for post-programming self-efficacy – pilot test

Items	Factor				
	General self-efficacy	**Independence and persistence**	**Complex programming tasks**	**Self-regulation**	**Simple programming tasks**
APSE1	.727				
APSE2	.736				
APSE3	.712				
APSE4			*.614*		
APSE5	.632				
APSE6	.650				
APSE7		.700			
APSE8		.782			
APSE9		.829			
APSE10		.593			
APSE11		.815			
APSE12			*.635*		
APSE13			*.740*		
APSE14			.801		
APSE15			.653		
APSE16			.645		
APSE17			*.556*	*.645*	
APSE18			.650		
APSE19				.862	
APSE20				.814	
APSE21			*.749*		
APSE22			*.728*		
APSE23			*.457*		
APSE24			*.494*		
APSE25			*.759*		

Kaiser-Meyer-Olkin Measure of Sampling Adequacy = .827 Bartlett's test of Sphericity Approx. Chi-square = 941.859 / df = 300 / Sig. = .000

Behavioural Engagement

The factor loadings of the 12 items in the behavioural engagement construct were examined and are presented in Table 6.36. A varimax rotation, with 3 factors on the behavioural engagement construct resulted in a KMO score that was > .5 (KMO = .584) and the significance of the Bartlett's test was < .05 (p = .000). All items loaded strongly on their intended construct and the total variance explained by the 3 factors was 56.72%. All item loadings were above the threshold of .5.

Table 6.36: Factor loadings for lower-order behavioural engagement constructs

Items	Factor		
	Effort	**Persistence**	**Help-seeking**
BEEF2	.640		
BEEF5	.778		
BEEF6	.624		
BEPA1	.716		
BEHS1			.760
BEHS3			.790
BEHS5			.693
BEPE1		.637	
BEPE3		.611	
BEPE4		.740	
BEPE5		.662	
BEEF3		.539	

Kaiser-Meyer-Olkin Measure of Sampling Adequacy = .584
Bartlett's test of Sphericity Approx. Chi-square = 181.841 / df = 66 / Sig. = .000

Cognitive Engagement

The factor loadings of the 15 items in the cognitive engagement construct were examined and are presented in Table 6.37. A varimax rotation, with 3 factors on the cognitive engagement construct resulted in a KMO score that was > .5 (KMO = .594) and the significance of the Bartlett's test was < .05 (p = .000). All items loaded strongly on their intended construct and the total variance explained by the 3 factors was 53.27%. Item loadings were above the threshold of .5. However, cross-loadings were observed on item CETE3. Item CETE3 appeared to be loading on two factors (*deep learning* and *trial and error*). Since the factor loading of item CETE3 was stronger on the *trial and error* construct, the item was retained in its intended construct.

Table 6.37: Factor loadings for lower-order cognitive engagement constructs

Items	Factor		
	Deep Learning	**Surface Learning**	**Trial and Error**
CEDL1	.526		
CEDL2	.758		
CEDL4	.632		
CEDL5	.660		
CESL2		.764	
CESL3		.576	
CESL5		.682	
CESL6		.614	
CESL8		.671	
CETE1			.594
CETE2			.775
CETE3	*.533*		.611
CETE4			.602
CETE5			.769
CESL4			.723

Kaiser-Meyer-Olkin Measure of Sampling Adequacy = .594
Bartlett's test of Sphericity Approx. Chi-square = 256.390 / df = 105 / Sig. = .000

Emotional Engagement

The factor loadings of the 11 items in the emotional engagement construct were examined and are presented in Table 6.38. A varimax rotation, with 3 factors on the emotional engagement construct resulted in a KMO score that was > .5 (KMO = .739) and the significance of the Bartlett's test was < .05 (p = .000). All items loaded strongly on their intended construct and the total variance explained by the 3 factors was 70.11%. Item loadings were above the threshold of .5. However, cross-loadings were observed on items EEGR3, and EEEN1. The item loading for EEEN1 was stronger (.1 difference) on the *enjoyment* construct than on the *interest* construct. The item was retained in the *enjoyment* construct, since the item had been validated in a previous research. The same can be observed with item EEGR3, which appears to load slightly stronger (.1 difference) on the *enjoyment* construct instead of the *gratification* construct. At this stage, EEGR3 was retained in the *gratification* construct since the items were derived from the findings of the focus groups. However, the factor loadings of item EEGR3 will be re-examined when the final set of data has been collected.

Table 6.38: Factor loadings for lower-order emotional engagement constructs

Items	Factor		
	Interest	**Gratification**	**Enjoyment**
EEEN1	*.511*		*.625*
EEEN2			.628
EEEN3			.761
EEGR1		.847	
EEGR2		.874	
EEGR3		*.516*	*.673*
EEGR4		.812	
EEGR5		.780	
EEIN1	.542		
EEIN3	.808		
EEIN4	.802		

Kaiser-Meyer-Olkin Measure of Sampling Adequacy = .739
Bartlett's test of Sphericity Approx. Chi-square = 307.232 / df = 55 / Sig. = .000

Programming Performance

The factor loadings of the 4 items in the programming performance construct were examined and are presented in Table 6.39. A varimax rotation, with 2 factors on the programming performance items resulted in a KMO score that was > .5 (KMO = .708) and the significance of the Bartlett's test was < .05 (p = .000). The total variance explained by the 2 factors was 86.75%. Item loadings were above the threshold of .5.

Table 6.39: Factor loadings for programming performance construct

Items	Factor	
	Self-assessment	**Programming Grade**
SA1	.650	
SA2	.807	
SA5	.935	
ProgGrade		.954

Kaiser-Meyer-Olkin Measure of Sampling Adequacy = .708
Bartlett's test of Sphericity Approx. Chi-square =
97.984 / df = 6 / Sig. = .000

6.4 Data Collection: Phase 1 and Phase 3

The actual data collection for Phase 1 took place between September 2013 and July 2014 while the data collection for Phase 3 took place from December 2013 to October 2014. The sample size, the unit of analysis, and the administration of the questionnaire are three factors that were considered during data collection.

6.4.1 Sample Size

Pinsonneault and Kraemer (1993) stressed that selecting a suitable sampling frame is critical in a survey research as the sample frame must sufficiently represent the population. In IS research, as a general rule, researchers who use SEM to analyse their data tend to use a lower boundary of 10 observations per indicator in determining the sample size (Chin, 1998a; Nunnally & Bernstein, 1994). Hinkin (1998) proposed that the indicator to observation ratio can range anywhere between 1:4 and 1:10 while Barlett, Kotrlik, and Higgins (2001) proposed that the item to response ratio can range between 1:5 and 1:10. If Hinkin's (1998) ratio is applied, the proposed sample size in this study should be between 268 and 670 (since there are 67 items - 25 items for programming self-efficacy, 38 items for engagement, 3 items for self-assessment, and 1 item for programming grade). On the other hand, Hair et al. (2014, p. 20) suggested that researchers who use the SEM-PLS data analysis technique could use Cohen's (1992) statistical power analysis to assess the minimum sample size. This study has a maximum of 9 arrows pointing to any one construct (programming grade or self-assessment). Thus, using Cohen's statistical power analysis, a minimum of 247 observations is needed to achieve a statistical power of 80% for detecting R^2 values of at least .1 and at a 1% probability of error. The data collection efforts were then focused on achieving a minimum of 247 observations.

6.4.2 Unit of Analysis

To ensure representativeness of the sample, the data was collected from two countries, Malaysia, and New Zealand. Both countries had local and international students enrolled in their introductory programming courses. The samples from these two countries were able to offer an international perspective on learning programming. A total of 28 HEIs were contacted to request access to the novice programmers in their introductory programming courses (Table 6.40). In New Zealand, the list of HEIs was obtained from The Computing and Information Technology Research and Education New Zealand (CITRENZ) website. This website contained a list of Universities and Polytechnics in New Zealand that offer Computer Science or Information Technology based courses. The response rate was poorer (15%) from the Polytechnics in New Zealand due to their policy on the privacy of student records. However, the response rate for the Universities in New Zealand was encouraging (75%). Only one University declined to participate since another research was being conducted on their students. Three other universities in New Zealand were not contacted since there were no introductory programming courses offered at the time of the data collection.

In Malaysia, a purposive sampling technique had to be employed to contact the HEIs due to time constraints, and the Malaysian government's strict enforcement of the Personal Data Protection Act 2010 (PDPA) which coincided with the start of the data collection in Malaysia. As some HEIs were unsure of how the PDPA would affect them, they declined to participate in the survey, citing confidentiality of student data as the main reason for doing so. Therefore, the HEIs were contacted through professional connections, and only 4 of the 9 HEIs contacted agreed to participate in the research.

Table 6.40: HEIs contacted

	New Zealand		Malaysia	
	Contacted	**Agreed**	**Contacted**	**Agreed**
Polytechnic	15	3	-	-
University	4	3	9	4
Total	**19**	**6**	**9**	**4**

6.4.3 Administration of Questionnaire

Phase 1 of the questionnaire was administered in Week 3 of the introductory programming course after receiving Human Ethics Approval. During the lecture, the researcher was given 15 minutes to brief the novice programmers about the purpose of the research, and to invite the novice programmers to participate in the research. The Participant Consent Form and the Participant Information Sheet was distributed to the novice programmers, and they were asked to return the Participant Consent Form to the researcher if they wished to participate in the research. When the researcher could not be present to brief the students, the course instructor acted as a proxy, who then briefed, and administered the questionnaire to the participants. To encourage the students to participate in the survey, the participants were entered into a draw which offered monetary rewards to the lucky winners.

The questionnaire was designed and administered online using Qualtrics software (Qualtrics, Provo, UT). Using the e-mail addresses on the Participant Consent Form, the participants were e-mailed the link to the survey questionnaire. The link to the questionnaire remained open for one week, and a reminder was sent to the participants to complete the questionnaire four days after the link was initially opened.

Like the survey in Phase 1, the survey in Phase 3 was also designed and administered online using the Qualtrics software (Qualtrics, Provo, UT). The survey was administered online after receiving Human Ethics Approval. An e-mail containing the link to the survey was sent to the participants who had participated in Phase 1 of the survey questionnaire. A copy of the Participant Information Sheet was attached to the e-mail. To encourage participation, the participants were reminded of the monetary incentive in the form of a lucky draw. The link to the questionnaire remained open for two weeks, and a reminder was sent to the participants to complete the questionnaire one week after the link was initially opened.

6.5 Data Preparation

Once the data collection phase had ended, the data was then prepared for analysis. The data was examined for missing values, the categorical variables were re-coded, and the data was examined for common method bias. SPSS (Version 20.0) was used to prepare the data for analysis.

6.5.1 Missing Values

Hair et al. (2014) proposed two guidelines for managing missing values. First, the percentage of missing values for each observation should not exceed 15% (Hair et al., 2014). This guideline was not an issue in the dataset, as the highest percentage of missing values for any one observation in the dataset did not exceed 2%. Second, if the percentage of missing values for an indicator in the dataset is less than 5%, then a mean value replacement strategy may be used when analysing the data in *SmartPLS 3* (Hair et al., 2014; Ringle, Wende, & Becker, 2014). Items BPSE18 and BPSE23 had the highest percentage of missing values in the dataset. The percentage of missing values for each indicator was 1.4%. Thus, the mean value replacement strategy was used in the analysis of the dataset in *SmartPLS 3*.

6.5.2 Handling Categorical Data

Since SEM-PLS does not handle categorical data efficiently (Hair et al., 2014), a continuous scale was developed for the categorical data in this study. The variables using categorical data in this study were intelligence and programming grade. Hair, Ringle and Sarstedt (2013) argue that using scales that do not have metric or quasi-metric data can violate the estimation of the PLS path model, and may result in misinterpretation in the analysis. Further, Hair et al. (2013) discouraged the use of dummy variables, particularly in reflective measurement models.

Table 6.41: Intelligence and programming grade categorical data as a continuous scale

	Intelligence & Programming Grade				
Categorical	E	D	C	B	A
Continuous	1 (Low Score)	2	3	4	5 (High Score)

The data collected for the intelligence construct and the programming grade construct are both categorical data. To overcome the limitation of using categorical data in the estimation of the PLS path model, Table 6.41 shows how intelligence and programming grade data may be measured on a continuous scale. Intelligence is measured by the school results of the student, and using the scale in Table 6.41, a poor grade (Grade E) implies a low score on a continuous scale, and a high grade (Grade A) implies a high score on a continuous scale. The same strategy was applied to the programming grade data.

6.5.3 Common Method Bias

Podsakoff, MacKenzie, Lee, and Podsakoff (2003) suggested that common method bias must be tested if the data for the independent and the dependent variables were not collected from more than one source, and if only a single method was used to gather data. Two statistical tests were performed to control for common method bias. The first test was an exploratory factor analysis (EFA) on all the measurement items. Podsakoff and Organ (1986) suggested that common method bias may exist in a dataset if the majority (>50%) of the items loaded on one factor. The results showed that the largest variance explained by an individual factor was 24.04%, suggesting that common method bias was not a problem in this study.

In the second test, the Harman's single factor test was performed. All the items were modelled as an indicator of a single factor. Here again, the single factor explained only 24.05% of the variance. Both these tests suggested that common method bias was not a problem in this study.

6.6 Chapter Summary

The scales for the Phase 1 and Phase 3 survey questionnaires were developed and examined for content, construct, and face validity. As a result of the validity assessments, participation was dropped from being an indicator of behavioural engagement due to a lack of reliable measures. The questionnaires were then pre-tested and a pilot study was conducted to improve the reliability of the survey questionnaires. The item-total correlation of the negatively worded items was weak during the reliability analysis of the pilot study data. Therefore, the negatively worded items were removed from the questionnaires. The sample size, the unit of analysis, and the administration of the questionnaire are three factors that were considered during the actual data collection. Finally, the data from the actual study was prepared for analysis using SPSS (Version 20.0). The data preparation included an analysis of the missing values, handling categorical data, and common method bias.

Chapter 7: Data Analysis (Survey)

Once the data from the Phase 1 and Phase 3 survey questionnaires were cleansed, the data was analysed, and the results were interpreted.

7.1 Data Analysis Strategy

The discussion of the results from the survey questionnaires will firstly show the demographics of the participants, their response rates, followed by a data normality test. Next, the data analysis technique - SEM-PLS is discussed, and the measurement model is examined to establish the reliability and validity of the constructs (Bagozzi, Yi, & Phillips, 1991). The measurement model describes the relationships between the constructs and their indicators (Gefen, Straub, & Boudreau, 2000; Hair et al., 2014). A factor analysis was performed to examine the convergent and discriminant validities. The factor analysis includes an exploratory factor analysis (EFA) and a Confirmatory Factor Analysis (CFA). Appendix H, Table AH.1 provides a summary of the validity and reliability tests that were performed in this study.

The hypotheses in the research model were then tested in the structural model. The structural model describes the relationships between the constructs (Gefen et al., 2000; Hair et al., 2014). The model was also examined for its predictive relevance and accuracy before the lower-order structural model was confirmed. The analysis of the measurement model and the structural model was based on the guidelines provided by Hair et al. (2014).

Finally, additional analyses were performed to examine the effect of programming self-efficacy on engagement, and the effect of engagement on the programming performance of the novice programmers. The additional analyses include the Importance Performance-Matrix Analysis (IPMA), tests on the significance of the confounding variables, higher-order constructs, multiple mediation, and multi-group analysis (MGA).

During the data analysis, the higher-order constructs *pre-* and *post-programming self-efficacy* was used to analyse the measurement model and the structural model after having satisfied all reliability and validity criteria for the measurement model. Appendix I contains the reliability and validity analysis of the lower-order constructs and the final measurement model of the lower-order programming self-efficacy constructs. Since the programming self-efficacy construct was established as a reflective-reflective type of model in Chapter 3, Section 3.4, all items of the lower-order constructs were repeated as items of the higher-order construct and the same validity criteria was applied to the higher-order constructs (Hair et al., 2014).

The terms endogenous and exogenous constructs are used throughout this chapter. These terms are used in SEM models. Endogenous constructs refer to the dependent variables in the model while exogenous constructs refer to the independent variables in the model (Hair et al., 2010).

7.2 Descriptive Statistics

Table 7.1 presents the response rate by country. Of the 1093 students approached to participate in the survey at the beginning of the course, 433 usable sets of responses (completed Phase 1 & Phase 3 survey) were received. The number of participants from Malaysia and New Zealand was almost equal, with 50.6% of the participants from Higher Educational Institutions (HEIs) in Malaysia, and 49.4% of the participants from HEIs in New Zealand.

In Phase 1 of the data collection, the overall response rate was 70.2% with a significantly lower response rate in New Zealand (59.6%) compared to Malaysia (93.3%). By Phase 3, 38.7% of the participants had dropped out, wherein 49.7% of the participants in New Zealand had dropped out in Phase 3 compared to 23.4% of participants that had dropped out in Malaysia.

Table 7.1 Response Rate by Country

Country	Sample Population	No. of Participants in Phase 1	No. of Participants in Phase 3	No. of Participants completed Phase 1 & 3
New Zealand	750 (68.6%)	447 (58.3%)	225 (47.9%)	214 (49.4%)
Malaysia	343 (31.4%)	320 (41.7%)	245 (52.1%)	219 (50.6%)
Total	**1093**	**767**	**470**	**433**

Table 7.2 presents the descriptive statistics of the participants. 72.7% of the participants were males. The significantly lower participation of females in programming related courses has been an on-going issue worldwide (Rubio, Romero-Zaliz, Mañoso, & de Madrid, 2015). 93.1% of the participants were less than 25 years old and close to half (46.9%) of the participants identified themselves as Asians which included participants from countries such as Malaysia, China, India, Indonesia, Pakistan, Vietnam, Philippines, Kazakhstan, and Uzbekistan. These participants were either studying in an HEI in Malaysia or in New Zealand. As can be seen from the ethnicity statistics, the two countries were able to offer an international perspective to this study, and a large number of the participants (88.9%) were from the Computer Science related majors.

Table 7.2 Descriptive Statistics of Participants

	Frequency (n = 433)			Percentage (%)
Gender				
Male	315			72.7
Female	118			27.3
Age				
25 years or less	403			93.1
More than 25 years	30			6.9
Ethnicity				
New Zealand European	131			30.3
Other European	5			1.2
New Zealander	18			4.2
Maori	5			1.2
Pacifica	3			.7
Asian	203			46.9
Middle Eastern	16			3.7
Latin American	3			.7
African	13			3.0
Others	36			8.3
Type of Major				
Major	385			88.9
Non-Major	48			11.1
Programming Grade Distribution				
	Malaysia	**New Zealand**	**Frequency (n = 433)**	
A	45	117	162	37.4
B	76	59	135	31.2
C	69	29	98	22.6
D	19	5	24	5.6
E	10	4	14	3.2

Table 7.2 also shows the grade distribution of the participants based on country. More than half of the participants (68.6%) had obtained at least a grade B in their introductory programming course while only 8.8% of the participants had obtained a grade D or lesser. Interestingly, a larger percentage of the participants in New Zealand (72.2%) obtained a Grade A, compared to the participants in Malaysia (27.8%).

In terms of programming experience, Table 7.3 shows that 66.7% of the participants did not have prior programming experience while of the 144 participants who did have programming experience, 90.2% of them had rated that their programming skills were at level 3 or less. This implies that most of the participants that had prior programming experience had between an average and a limited ability to program. For analysis purposes, the answers to the question of how did you learn programming was re-coded into three categories and is presented in Table 7.3. 68.7% of the participants with programming experience had learned programming by attending a course that was offered in their school, or by attending a training programme.

Table 7.3 Programming Experience

Programming Experience		
Yes	144	33.3
No	289	66.7
Level of Programming Experience		
	Frequency (n = 144)	**Percentage (%)**
1 (Limited Programming Experience)	32	22.2
2	52	36.1
3	46	31.9
4	12	8.3
5 (Extensive Programming Experience)	2	1.4
How did you learn Programming?		
	Frequency	**Percent**
Attended Course/Training	99	68.7
Learned Programming On My Own	43	29.9
At My Workplace	2	1.4

7.3 Test for Normality

Like the pilot study, the data was tested for normality by assessing the skewness and kurtosis values. The skewness and kurtosis values should be close to zero (0) (Field, 2013; Hair et al., 2014), and may be within an acceptable range of +/- 2 (Gravetter & Wallnau, 2014). Table 7.4 displays the skewness and kurtosis values of each construct in the research model. The skewness of the constructs was between -1.463 and .611 while the kurtosis of the constructs was between -.960 and 1.835. These values are within the acceptable range of +/- 2, confirming that the data was normally distributed.

Table 7.4: Skewness and Kurtosis - Distribution of Data

Higher-Order Construct	Lower-order Construct	Skewness		Kurtosis	
		Statistic	Std. Error	Statistic	Std. Error
Pre-Programming Self-Efficacy	General self-efficacy	-.390	.117	-.242	.234
	Independence and persistence	-.506		-.232	
	Complex programming tasks	-.280		-.251	
	Self-regulation	-.330		-.249	
	Simple programming tasks	-.369		-.155	
Post-Programming Self-Efficacy	General self-efficacy	-.458		.029	
	Independence and persistence	-.722		.860	
	Complex programming tasks	-.210		-.269	
	Self-regulation	-.354		-.186	
	Simple programming tasks	-.397		-.077	
Behavioural Engagement	Help Seeking	-.141		-.112	
	Effort	-.112		-.298	
	Persistence	-.348		-.158	
Cognitive Engagement	Deep Learning	-.396		.246	
	Surface Learning	.101		.035	
	Trial and Error	.611		-.216	
Emotional Engagement	Interest	-.431		-.130	
	Gratification	-.685		.094	
	Enjoyment	-1.463		1.835	
	Self-Assessment	-.408		-.810	
	Programming Grade	-.361		-.960	

7.4 Structural Equation Modeling (SEM)

The data collected from Phase 1 and Phase 3 of this study were analysed using a Structural Equation Modeling (SEM) technique. SEM is a second generation multivariate analysis technique that is popular with researchers. SEM overcomes the weaknesses of first generation techniques such as principal component and linear regression analysis (Hair et al., 2014). SEM statistical models aim to test the research hypotheses that were developed from theory by examining the relationships between the constructs, the direction of the relationships and their significance (Hair et al., 2010; Schreiber et al., 2006). Chin (1998) argues that SEM is preferred over other techniques such as principal components analysis, factor analysis, discriminant analysis, or multiple regressions, as SEM offers flexibility in the interplay between theory and data. Additionally, the SEM technique is proposed in this study

as it enables flexibility in the modelling of multiple predictor and criterion variables (Chin, 1998).

There are two types of SEM techniques. The covariance-based SEM (CB-SEM) is used primarily for hypothesis testing by confirming or rejecting theories while the variance-based partial least squares (PLS-SEM) is used primarily for exploratory research and to develop theories (Hair et al., 2014, p. 4). The variance-based PLS-SEM was used to analyse the quantitative data in this study. This is because PLS-SEM is appropriate when the objective of the research is to predict and explain the constructs, and when the theory is not well established (Hair et al., 2014, p. 14). Further, PLS-SEM is able to handle complex models, is robust for smaller sample sizes, handles reflective and formative measures, and can handle single-item constructs (Hair et al., 2014).

7.5 Measurement Model

The first step in analysing the measurement model was to perform an EFA on the dataset. The EFA was examined using SPSS (Version 20.0). Next, using SmartPLS 3 (Ringle et al., 2014), a CFA was performed on the measurement model. The factor loadings of the indicators were examined for their reliability and validity. The path connecting the items to the constructs was measured reflectively. The internal consistency (Composite Reliability – CR), Cronbach's alpha estimate, indicator reliability, convergent validity (Average Variance Extracted – AVE), and discriminant validity were established for the measurement model.

7.5.1 Exploratory Factor Analysis (EFA)

The factors that were considered for the EFA during the pilot study were used in the EFA for the measurement model (Chapter 6, Section 6.3.2.7). The same assessment criteria to analyse the factors was used, with the exception of the factor loadings. Since the sample size was more than 350, a factor loading of .3 was considered sufficient for significance at the .05 level (Hair et al., 2010, p. 117).

The EFA was performed between the programming self-efficacy construct and the lower-order engagement constructs, and between the lower-order engagement constructs and the programming performance construct. Since the programming self-efficacy scale had been validated previously, and an EFA was performed during the pilot study with reasonably strong factor loadings, the EFA was not repeated at this stage.

7.5.1.1 Behavioural Engagement

The factor loadings of the 12 items in the behavioural engagement construct were examined and are presented in Table 7.5. A varimax rotation with 3 factors resulted in a KMO score that was > 0.5 (KMO = .796) and the significance of the Bartlett's test was < .05 (p = .000). All items loaded strongly on their intended construct with the exception of the cross-loadings of two items. Item BEPE1 loaded strongly on the *effort* construct. When an oblimin rotation was performed, item BEPE1 had still loaded strongly on the *effort* construct. However, since the difference in the factor loading was small (.156), and after reviewing the relevance of the item within the *persistence* construct, the item was retained in the persistence construct. Next, item BEPE5 had acceptable factor loadings on the *effort* and *persistence* construct. However, since the factor loading of item BEPE5 was stronger in the *persistence* construct, this item was retained in the persistence construct. The total variance explained by the 3 factors was 56.56%. All item loadings were above the threshold of .3.

Table 7.5: Factor loadings for lower-order behavioural engagement constructs

Items	**Factor**		
	Effort	**Persistence**	**Help-seeking**
BEEF2	.802		
BEEF5	.541		
BEEF6	.748		
BEPA1	.540		
BEHS1			.800
BEHS3			.678
BEHS5			.783
BEPE1	*.583*	*.427*	
BEPE3		.734	
BEPE4		.658	
BEPE5	*.487*	*.607*	
BEEF3		.775	

Kaiser-Meyer-Olkin Measure of Sampling Adequacy = .796
Bartlett's test of Sphericity Approx. Chi-square = 1361.733 / df = 66 / Sig. = .000

7.5.1.2 Cognitive Engagement

The factor loadings of the 15 items in the cognitive engagement construct were examined and are presented in Table 7.6. A varimax rotation with 3 factors resulted in a KMO score that was > 0.5 (KMO = .839) and the significance of the Bartlett's test was < .05 (p = .000). All the items loaded strongly on their intended construct with the exception of two items. Item CETE1 had loaded on the *deep learning* construct instead of the *trial and error* construct. When an oblimin rotation was performed, the item CETE1 continued to load strongly on the *deep learning* construct. Thus, CETE1 was re-assigned as a measure for the *deep learning* construct. Item CETE4 was loading strongly on the *deep learning* and the *trial and error* constructs with only a .045 difference in the loadings. An oblimin rotation on the item produced the same factor loadings. This suggested that CETE4 was ambiguous and the decision was made to delete the item. The total variance explained by the 3 factors was 51.49%. All item loadings were above the threshold of .3.

Table 7.6: Factor loadings for lower-order cognitive engagement constructs

Items	Factor		
	Deep Learning	**Surface Learning**	**Trial and Error**
CEDL1	.740		
CEDL2	.613		
CEDL4	.667		
CEDL5	.599		
CESL2		.708	
CESL3		.704	
CESL5		.688	
CESL6		.630	
CESL8		.629	
CETE1	*.654*		
CETE2			.576
CETE3			.745
CETE4	*.540*		*.495*
CETE5			.727
CESL4			.699

Kaiser-Meyer-Olkin Measure of Sampling Adequacy = .839
Bartlett's test of Sphericity Approx. Chi-square = 1651.001 / df = 105 / Sig. = .000

7.5.1.3 Emotional Engagement

The factor loadings of the 11 items in the emotional engagement construct were examined and are presented in Table 7.7. A varimax rotation with 3 factors resulted in a KMO score that was > 0.5 (KMO = .903) and the significance of the Bartlett's test was < .05 (p = .000). All items loaded strongly on their intended construct with the exception of two items. Item EEEN3 loaded strongly on the *interest* and the *enjoyment* constructs. The item was retained in the *enjoyment* construct, since it had been validated as an indicator of enjoyment in previous research. Next, item EEGR3 loaded strongly on the *gratification* and the *enjoyment* constructs with a .043 difference in the loadings. An oblimin rotation on the item produced a similar outcome. This suggested that EEGR3 was ambiguous and the decision was made to delete the item. The total variance explained by the 3 factors was 67.82%. All item loadings were above the threshold of .3.

Table 7.7: Factor loadings for lower-order emotional engagement constructs

Items	Factor		
	Interest	**Gratification**	**Enjoyment**
EEEN1			.675
EEEN2			.823
EEEN3	*.448*		*.663*
EEGR1		.764	
EEGR2		807	
EEGR3		*.532*	*.489*
EEGR4		.628	
EEGR5		.780	
EEIN1	.813		
EEIN3	.608		
EEIN4	.763		

Kaiser-Meyer-Olkin Measure of Sampling Adequacy = .903
Bartlett's test of Sphericity Approx. Chi-square = 2086.251 / df = 55 / Sig. = .000

7.5.1.4 Programming Performance

The factor loadings of the 4 items in the programming performance scale were examined and are presented in Table 7.8. A varimax rotation with 2 factors resulted in a KMO score that was > 0.5 (KMO = .680) and the significance of the Bartlett's test was < .05 (p = .000). All items loaded strongly on their intended construct with the exception of item SA1. Item SA1 loaded strongly on the *self-assessment* and the *programming grade* constructs. Since *self-assessment* is a self-reported measure and *programming grade* is an objective measure of programming performance, the decision was made to retain item SA1 in the *self-assessment* construct. The total variance explained by the 2 factors was 77.87%. Item loadings were above the threshold of .5.

Table 7.8: Factor loadings for programming performance constructs

Items	Factor	
	Self-assessment	**Programming Grade**
SA1	*.696*	*.524*
SA2	.809	
SA5	.844	
ProgGrade		.942

Kaiser-Meyer-Olkin Measure of Sampling Adequacy = .680
Bartlett's test of Sphericity Approx. Chi-square = 443.364 / df = 6 / Sig. = .000

7.5.2 Reliability

Once the EFA was performed on the measurement model, the next step was to perform a CFA on the measurement model. In the CFA, the internal consistency reliability and the validity of the measurement model were assessed. The composite reliability (CR), indicator reliability, and Cronbach's α (alpha) are three estimates for internal consistency reliability in SEM-PLS (Hair et al., 2014). Hair et al. (2014) suggest that the reliability of the constructs could be assessed using the composite reliability (CR) instead of Cronbach's α as the latter delivers a conservative estimate of the internal consistency reliability. Nevertheless, both the CR and Cronbach α estimates of the indicator variables were estimated and are presented in Table 7.10.

A CR estimate of > .7 is acceptable particularly in confirmatory research (Kline, 1999; Moore & Benbasat, 1991; Nunnally & Bernstein, 1994). However, a CR estimate that is > .95 should be avoided since a value with high reliability suggests that there are redundant indicators in the scale (Hair et al., 2014). Based on Table 7.10, all the CR estimates of each of the constructs in the measurement model meet the acceptable threshold of > .7. The higher-order *post-programming self-efficacy* construct had a CR estimate of .956 which is marginally higher than the estimate that suggests that the constructs may have redundant indicators. Despite the high CR estimate, the items in the *post-programming self-efficacy* construct were retained for two reasons. First, the *post-programming self-efficacy* construct is a higher-order construct, and second, the CR estimate (.956) is marginally higher that the suggested CR estimate of .95.

The Cronbach's α for the constructs *help-seeking* and *surface learning* were > .6. A Cronbach's α estimate of .6 is acceptable in psychological constructs due to its diversity and exploratory nature (Kline, 1999; Nunally, 1978; Nunnally & Bernstein, 1994) and generally indicates fair reliability (Zikmund et al., 2010). Since, the Cronbach's α for *surface learning* was close to the acceptable threshold of .7, and the CR of *help-seeking* and *surface learning* were above > .7, these two constructs were retained in the measurement model.

The assessment of the indicator reliability is discussed in Section 7.5.3.

7.5.3 Convergent Validity

Convergent validity examines how well the items that measure a construct correlate (Bagozzi & Phillips, 1982; Hair et al., 2014). In SEM-PLS, the average variance extracted (AVE) and the outer loadings of the indicators are examined when assessing the convergent validity of the measurement model. An outer loading of ≥ *.7* indicates that the indicator loads well onto the construct while outer loadings that are between *.4* and *.7* are typically examined for the AVE and CR before a decision is made to retain or delete the indicators (Hair et al., 2010; Hair et al., 2014). If the deletion of the item results in an increase in the AVE and CR, then the item should be deleted. Any item with an outer loading that is < *.40* is recommended to be deleted (Hair et al., 2014).

Table 7.9 lists the items that had weak outer loadings, and that were deleted in order to improve the AVE value. The AVE is the "grand mean value of the squared loadings of the indicators associated with the construct… and is referred to as the communality of the construct" (Hair et al., 2014, p. 103). An AVE value of > *.5* is acceptable, as this implies that the construct explains more than half of the variance of its indicators (Hair et al., 2014). Eleven items were deleted from the *pre-programming self-efficacy* construct in order to improve the AVE from .380 to .505 while six items were deleted from the *post-programming self-efficacy* construct to improve the AVE from .476 to .535. In addition, one item (CESL8) was deleted from the *surface learning* construct in order to improve the AVE from .452 to .515.

Table 7.9: Indicators deleted to improve AVE

Construct	Items Deleted	To improve AVE	
		From	To
Pre-Programming Self-Efficacy	BPSE9 BPSE8 BPSE11 BPSE10 BPSE20 BPSE7 BPSE19 BPSE3 BPSE5 BPSE2 BPSE4	.380	.505
Post-Programming Self-Efficacy	APSE11 APSE8 APSE9 APSE19 APSE10 APSE3	.476	535
Surface Learning	CESL8	.452	.515

Table 7.10 lists the final set of constructs, the outer loadings of the items, and the AVE of the constructs after meeting the conditions for convergent validity. Although there were items with an outer loading of < *.7*, these items were retained since they did not improve the AVE scores when the item(s) were deleted. All conditions to satisfy the convergent validity criteria have thus been met.

Table 7.10: Convergent Validity of Measurement Model

Construct	Indicator	Loadings	AVE	CR	Cronbach's α
Pre-Programming Self-Efficacy	BPSE1	.627	.505	.934	.924
	BPSE6	.649			
	BPSE12	.673			
	BPSE13	.765			
	BPSE14	.779			
	BPSE15	.760			
	BPSE16	.646			
	BPSE17	.711			
	BPSE18	.725			
	BPSE21	.725			
	BPSE22	.759			
	BPSE23	.682			
	BPSE24	.742			
	BPSE25	.679			
Post Programming Self-Efficacy	APSE1	.758	.535	.956	.951
	APSE2	.655			
	APSE4	.674			
	APSE5	.763			
	APSE6	.760			
	APSE7	.647			
	APSE12	.740			
	APSE13	.781			
	APSE14	.778			
	APSE15	.765			
	APSE16	.714			
	APSE17	.766			
	APSE18	.755			
	APSE20	.651			
	APSE21	.710			
	APSE22	.766			
	APSE23	.694			
	APSE24	.750			
	APSE25	.749			

Construct	Indicator	Loadings	AVE	CR	Cronbach's α
Effort	BEEF2	.792	.517	.808	.701
	BEEF5	.568			
	BEEF6	.707			
	BEPA1	.787			
Help-Seeking	BEHS1	.868	.535	.762	.659
	BEHS3	.424			
	BEHS5	.820			
Persistence	BEPE1	.726	.519	.842	.767
	BEPE3	.568			
	BEPE4	.749			
	BEPE5	.832			
	BEEF3	.702			
Deep Learning	CEDL1	.766	.508	.837	.757
	CEDL2	.638			
	CEDL4	.743			
	CEDL5	.773			
	CETE1	.638			
Surface Learning	CESL2	.784	.515	.805	.695
	CESL3	.695			
	CESL5	.827			
	CESL6	.527			
Trial and Error	CETE2	.547	.548	.825	.723
	CETE3	.827			
	CETE5	.695			
	CESL4	.853			
Enjoyment	EEEN1	.811	.625	.832	.701
	EEEN2	.685			
	EEEN3	.865			
Gratification	EEGR1	.849	.592	.851	.768
	EEGR2	.637			
	EEGR4	.685			
	EEGR5	.878			
Interest	EEIN1	.699	.677	.862	.766
	EEIN3	.885			
	EEIN4	.872			
Self-Assessment	SA1	.874	.677	.862	.759
	SA2	.871			
	SA5	.714			
*__Programming Grade__	ProgGrade	1.000	1.000	1.000	1.000

*_single-item construct_

7.5.4 Discriminant Validity

Discriminant validity refers to how well a construct differs from the other constructs in the model through statistical comparisons (Bagozzi & Phillips, 1982; Hair et al., 2014). There are two conditions to determine the discriminant validity of the constructs.

The first condition states that the indicators should load more strongly on their construct than on other constructs (Hair, Ringle, & Sarstedt, 2011). The cross-loading of the items was examined and is presented in Appendix H, Table AH.2[6]. The outer loadings of the items that are highlighted in grey confirm that the items load strongly on their intended construct than on other constructs.

The second condition applies the Fornell-Larcker criterion which states that the square root of a construct's AVE should be higher than any of the correlations with other constructs (Fornell & Larcker, 1981). Table 7.11 presents the results using the Fornell-Larcker criterion. The diagonal values that are highlighted in the bold text represent the square root of the AVE while the off-diagonals represent the correlations. The results of the Fornell-Larcker criterion confirm that the square root of a construct's AVE is higher than the correlations of the other constructs. Thus, all conditions to satisfy the discriminant validity of the measurement model have been met.

[6] The table containing the cross-loadings was moved to the Appendix as the table spans across 3 pages.

Table 7.11: Fornell-Larcker criterion for discriminant validity of measurement model

Construct	Deep Learning	Effort	Enjoyment	Gratification	Help Seeking	Interest	Persistence	Post-programming Self-Efficacy	Pre-programming Self-Efficacy	Programming Grade	Self-assessment	Surface Learning	Trial and Error
Deep Learning	**0.713**												
Effort	0.495	**0.719**											
Enjoyment	0.585	0.454	**0.791**										
Gratification	0.607	0.361	0.537	**0.769**									
Help Seeking	-0.042	0.113	-0.142	-0.007	**0.732**								
Interest	0.505	0.538	0.693	0.518	-0.021	**0.823**							
Persistence	0.652	0.561	0.561	0.524	0.005	0.549	**0.721**						
Post-programming Self-efficacy	0.676	0.560	0.625	0.470	-0.192	0.564	0.633	**0.732**					
Pre-programming Self-Efficacy	0.407	0.287	0.383	0.198	-0.113	0.300	0.364	0.568	**0.710**				
Programming Grade	0.271	0.230	0.314	0.140	-0.287	0.166	0.207	0.416	0.282	**1.000**			
Self-assessment	0.573	0.534	0.559	0.501	-0.108	0.529	0.525	0.695	0.388	0.355	**0.823**		
Surface Learning	-0.140	0.059	-0.181	-0.094	0.516	-0.032	-0.086	-0.273	-0.223	-0.354	-0.175	**0.718**	
Trial and Error	0.560	0.550	0.450	0.452	0.061	0.412	0.611	0.516	0.329	0.133	0.459	0.038	**0.741**

Note: Diagonals represent the square root of the AVE while the off-diagonals represent the correlations

7.5.5 Final Measurement Model

Figures 7.1, 7.2, and 7.3 present the final measurement model for the research model after performing the EFA, and after satisfying the conditions for convergent and discriminant validity. For ease of illustration, the model has been divided into the three engagement constructs (behavioural engagement, cognitive engagement, and emotional engagement).

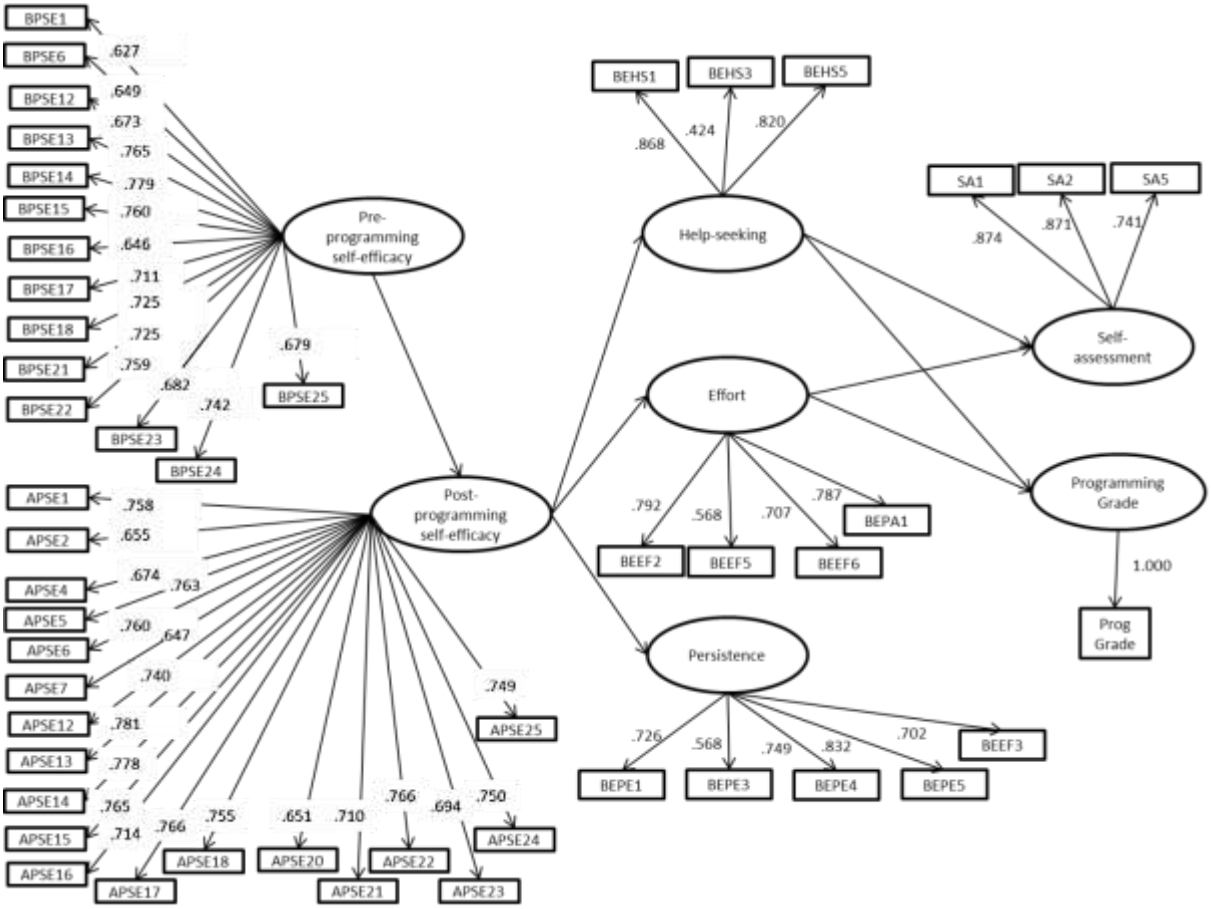


Figure 7.1: Measurement model with lower-order behavioural engagement constructs

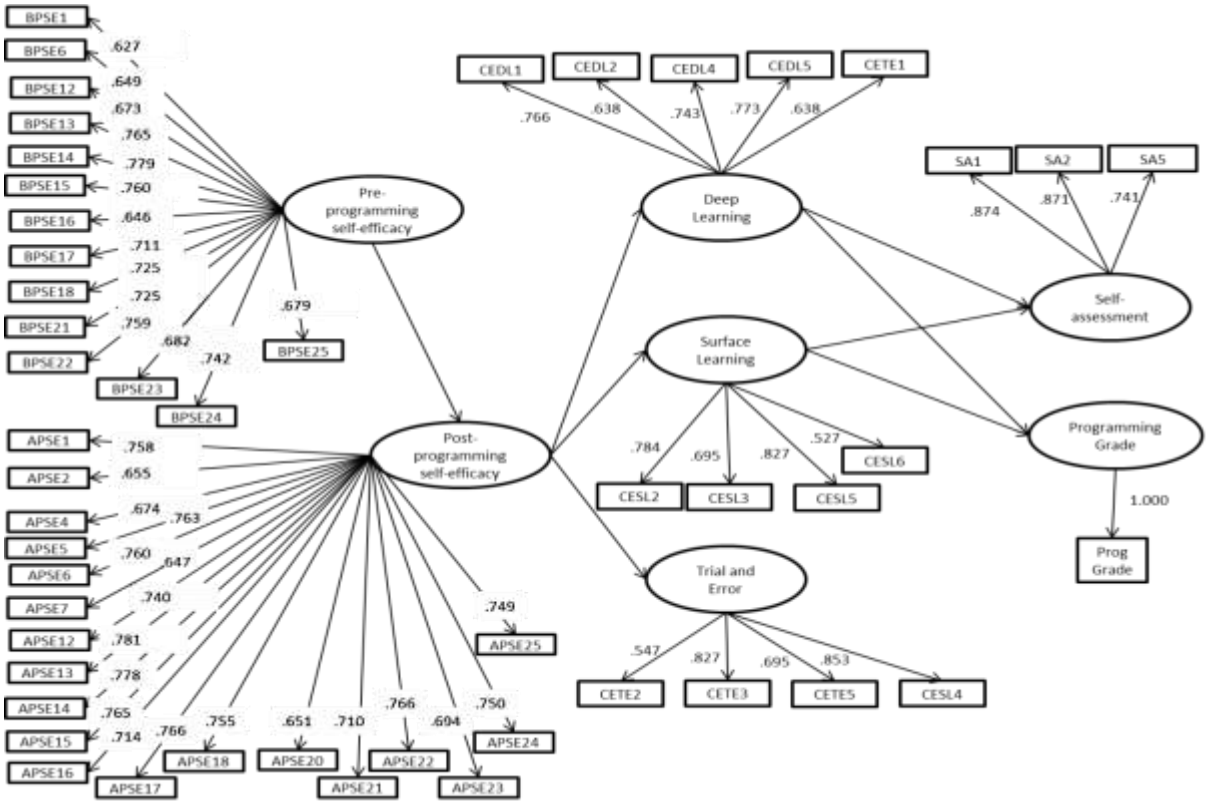


Figure 7.2: Measurement model with lower-order cognitive engagement constructs

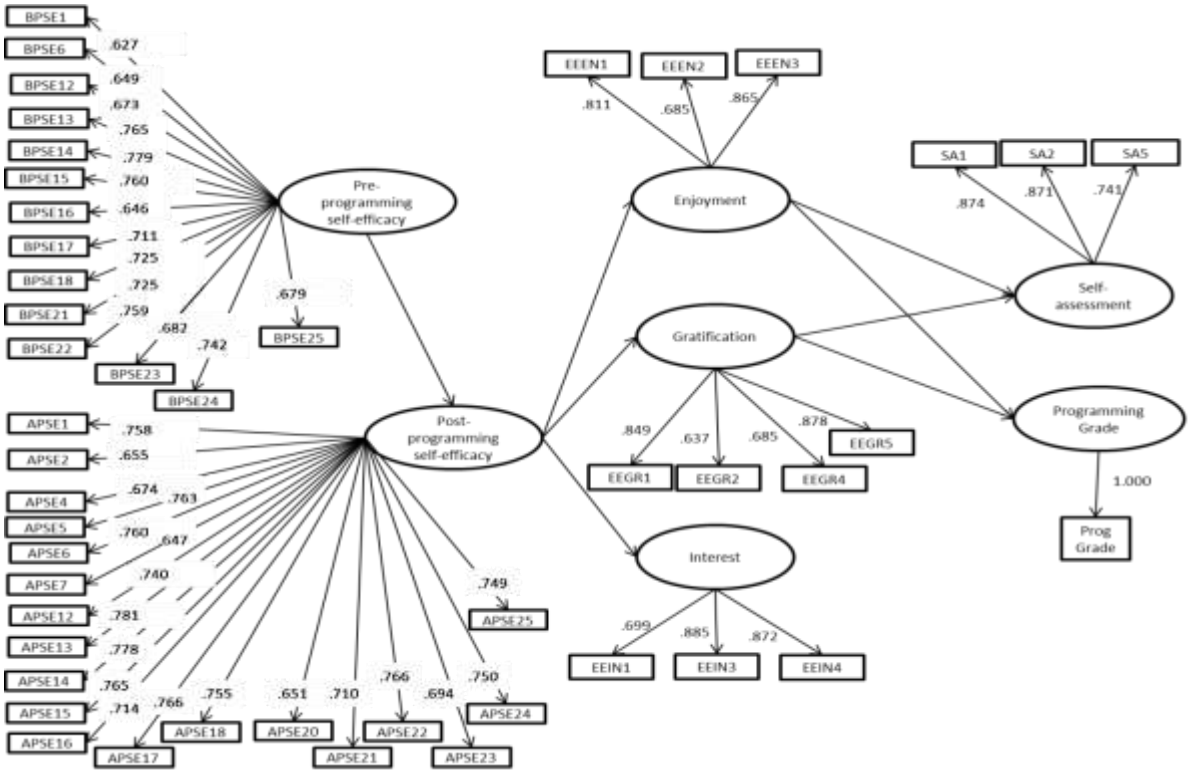


Figure 7.3: Measurement model with lower-order emotional engagement constructs

7.6 Structural Model

Once the conditions to validate the measurement model were met, the structural model was then examined for the relationships between the constructs and to assess the model's predictive capability. Hair et al. (2014, p. 169) suggested the following steps for the evaluation of the structural model:

Step 1: Assess the structural model for collinearity issues

Step2: Assess the significance and relevance of the structural model relationships

Step 3: Assess the level of R^2

Step 4: Assess the effect sizes of f^2

Step 5: Assess the predictive relevance Q^2 and the q^2 effect sizes

7.6.1.1 Collinearity Issues

In the first step, the structural model was examined for collinearity, which refers to high correlations between the constructs (Hair et al., 2014). High collinearity may impact the structural model as the error rates may increase, resulting in an inaccurate estimation of the loadings and the sign being reversed (Hair et al., 2014). To assess collinearity, the tolerance and VIF (Variance Inflation Factor) was computed by performing a multiple regression analysis on the predictor variables (lower-order engagement constructs). The post-programming self-efficacy construct was not examined for collinearity since the construct had only one predictor variable. Tolerance levels of $\leq .2$ and a VIF value of ≥ 5 indicate a potential collinearity problem (Hair et al., 2011). Table 7.12 displays the tolerance values and VIF values of the predictor variables. All the engagement constructs had a tolerance level of > 0.2 and a VIF of < 5. This indicates that there are no collinearity issues between each set of the engagement constructs (predictor variables).

Table 7.12: Collinearity of predictor variables

Predictor Variables	Programming Grade		Self-Assessment	
	Collinearity Statistics			
	Tolerance	VIF	Tolerance	VIF
Effort	.502	1.993	.502	1.993
Persistence	.412	2.429	.412	2.429
Help-seeking	.694	1.441	.694	1.441
Deep Learning	.381	2.627	.381	2.627
Surface Learning	.657	1.521	.657	1.521
Trial and Error	.513	1.950	.513	1.950
Enjoyment	.341	2.929	.341	2.929
Gratification	.535	1.871	.535	1.871
Interest	.363	2.759	.363	2.759

7.6.1.2 Hypothesis Testing using Bootstrapping

The bootstrapping procedure was used to determine the significance of the path coefficients (Chin, 1998a). Since PLS-SEM assumes that the data is not normally distributed, a nonparametric bootstrapping procedure would be more appropriate to determine the significance of the coefficients (Davison & Hinkley, 1997). The bootstrapping procedure "estimates the standard errors of the parameter estimates, calculates the ratio of a parameter estimate to its standard error, and compares this statistic to the *t* distribution to obtain the *p*-value" (Rönkkö & Evermann, 2013, p. 15).

Five thousand (5000) bootstrap samples were used to estimate the PLS path model (Hair et al., 2014). Table 7.13 presents the outcome of the hypothesis testing using the bootstrapping procedure. Path coefficients that are > *.20* are significant while path coefficients that are < *.1* are not significant (Hair et al., 2014, p. 86). Lohmöller (1989) suggested that path coefficients that are > *.1* are acceptable while Chin (1998a) suggested that path coefficients that are between *.20* and *.30* are acceptable. Additionally, path coefficients that are close to *+1* have strong positive relationships and are statistically significant (the opposite is true for negative relationships) (Hair et al., 2014).

In the first run of the bootstrap sample, the "no sign change" option was applied followed by a "construct level sign change" option. According to Hair et al. (2014) the "no sign change" option is conservative and the "construct level sign change" option may be applied to improve the significance of the path coefficients. There were no significant differences between the "no sign change" option and the "construct level sign change" option in this study. Table 7.13 presents the results of the bootstrapping test, the significance of the hypothesis tests using bootstrapping are discussed and the final structural model is then presented in Section 7.6.1.7 (Figures 7.4, 7.5 and 7.6).

Relationship between pre-programming self-efficacy and post programming self-efficacy

The relationship between *pre-programming self-efficacy* and *post-programming self-efficacy* was significant at a 1% probability of error. Hypothesis H1 *(β=.568; t=15.107)* was supported and there was a positive relationship between *pre-programming self-efficacy* and *post-programming self-efficacy*.

Relationship between post-programming self-efficacy and lower-order engagement constructs

All the relationships between *post-programming self-efficacy* and the engagement constructs were significant at a 1% probability of error. Hypotheses H4 *(β=.560; t=16.590)*, H5 *(β=.633; t=19,377)*, H6 *(β=.676; t=24.052)*, H8 *(β=.516; t=12.591)*, H10 *(β=.625; t=22.868)*, H20 *(β=.470; t=11.745)*, and H9 *(β=.564; t=18.055)* were supported with a positive relationship observed between *post-programming self-efficacy* and the lower-order

engagement constructs. Hypothesis H7 *(β=-.273; t=5.221)* was supported by a negative relationship observed between *post-programming self-efficacy* and *surface learning* and was significant at a 1% probability of error.

However, Hypothesis H3 *(β=-.192; t=3.221)* showed a negative relationship between *post-programming self-efficacy* and *help-seeking* and was significant at a 1% probability of error. The path coefficient between *post-programming self-efficacy* and *help-seeking* had a weak significance (path coefficient <-.*2*). Since hypothesis H3 was initially hypothesised as a positive relationship, this relationship is thus not supported.

Relationship between lower-order engagement constructs and programming grade

Behavioural Engagement

Hypothesis H13a *(β=.230; t=3.910)* was supported with a positive relationship observed between *effort* and *programming grade* and was significant at a 1% probability of error.

Hypothesis H12a *(β=-.149; t=2.993)* showed a negative relationship between *help-seeking* and *programming grade* and was significant at a 1% probability of error. However, the path coefficient between *help-seeking* and *programming grade* had a weak significance (path coefficient <-.*2*). Since hypothesis H12a was initially hypothesised as a positive relationship, this relationship was not supported. The relationship between *persistence* and *programming grade* (H14a) *(β=-.011; t=.162)* was not supported since it was not statistically significant.

Cognitive Engagement

Hypothesis H15a *(β=.129; t=2.007)* was supported by a positive relationship observed between *deep learning* and *programming grade* and was significant at a 5% probability of error. The path coefficient for hypothesis H15a had weak significance (path coefficient <.*2*). Hypothesis H16a *(β=-.235; t=4.495)* was supported by a negative relationship observed between *surface learning* and *programming grade* and was significant at a 1% probability of error.

On the other hand, the relationship between *trial and error* (H17a) *(β=-.062; t=1.006)* and *programming grade* were not supported since it was not statistically significant.

Emotional Engagement

Hypothesis H19a *(β=.266; t=3.713)* was supported by a positive relationship observed between *enjoyment* and *programming grade* and was significant at a 1% probability of error.

Hypothesis H18a *(β=-.167; t=2.510)* showed a negative relationship between *interest* and *programming grade* and was significant at a 5% probability of error. However, the path coefficient between *interest* and *programming grade* had a weak significance (path coefficient <-.*2*). Since hypothesis H18a was initially hypothesised as a positive relationship,

this relationship was not supported. The relationship between *gratification* (H21a) *(β=-.068; t=1.181)* and *programming grade* was also not supported since it was not statistically significant.

Relationship between lower-order engagement constructs and self-assessment

Behavioural Engagement

Hypothesis H13b *(β=.257; t=5.112)* was supported by a positive relationship observed between *effort* and *self-assessment* and was significant at a 1% probability of error.

The relationship between *persistence* and *self-assessment* (H14b) *(β=.034; t=.623)*, and the relationship between *help-seeking* and *self-assessment* (H12b) *(β=-.063; t=1.402)* were not supported since both the relationships were not statistically significant.

Cognitive Engagement

Hypothesis H15b *(β=.175; t=2.941)* was supported by a positive relationship observed between *deep learning* and *self-assessment* and was significant at a 1% probability of error. However, the path coefficient for hypothesis H15b had weak significance (path coefficient *<.2*). Hypothesis H16b *(β=-.091; t=1.863)* was supported by a negative relationship observed between *surface learning* and *self-assessment* and was significant at a 10% probability of error. However, the path coefficient between *surface learning* and *self-assessment* had a weak significance (path coefficient *<-.2*).

On the other hand, the relationship between *trial and error* (H17b) *(β=.047; t=.877)* and *self-assessment* were not supported since it was not statistically significant.

Emotional Engagement

Hypothesis H19b *(β=.136; t=2.287)* was supported by a positive relationship observed between *enjoyment* and *self-assessment* and was significant at a 5% probability of error. Similarly, hypothesis H21b *(β=.136; t=2.332)* was supported by a positive relationship between *gratification* and *self-assessment* and was significant at a 5% probability of error.

On the other hand, the relationship between *interest* (H18b) *(β=.088; t=1.449)* and *self-assessment* were not supported since it was not statistically significant.

Table 7.13: Hypothesis testing using Bootstrapping (with lower-order engagement constructs)

Hypothesis	Relationship	Path Coefficient	Standard Error	t-value	*p*-value	Decision
H1	Pre-programming self-efficacy -> Post-programming self-efficacy	.568	.038	15.107***	.000	Supported
H3	*Post-programming self-efficacy -> Help-seeking*	*-.192*	*.060*	*3.221***	*.001*	*Not Supported*
H4	Post-programming self-efficacy -> Effort	.560	.034	16.590***	.000	Supported
H5	Post-programming self-efficacy -> Persistence	.633	.033	19.377***	.000	Supported
H6	Post-programming self-efficacy -> Deep Learning	.676	.028	24.052***	.000	Supported
H7	Post-programming self-efficacy -> Surface Learning	-.273	.052	5.221***	.000	Supported
H8	Post-programming self-efficacy -> Trial and Error	.516	.041	12.591***	.000	Supported
H9	Post-programming self-efficacy -> Interest	.564	.031	18.055***	.000	Supported
H10	Post-programming self-efficacy -> Enjoyment	.625	.027	22.868***	.000	Supported
H20	Post-programming self-efficacy -> Gratification	.470	.040	11.745***	.000	Supported
H12a	*Help-seeking -> Programming Grade*	*-.149*	*.050*	*2.993***	*.003*	*Not Supported*
H13a	Effort -> Programming Grade	.230	.059	3.910***	.000	Supported
H14a	*Persistence -> Programming Grade*	*-.011*	*.067*	*0.162*	*.871*	*Not Supported*
H15a	Deep Learning -> Programming Grade	.129	.066	2.007**	.050	Supported
H16a	Surface Learning -> Programming Grade	-.235	.052	4.495***	.000	Supported

Hypothesis	Relationship	Path Coefficient	Standard Error	t-value	*p*-value	Decision
H17a	*Trial and Error -> Programming Grade*	*-.062*	*.061*	*1.006*	*.315*	*Not Supported*
H18a	*Interest -> Programming Grade*	*-.167*	*.067*	*2.510***	*.012*	*Not Supported*
H19a	Enjoyment -> Programming Grade	.266	.072	3.713***	.000	Supported
H21a	*Gratification -> Programming Grade*	*-.068*	*.057*	*1.181*	*.238*	*Not Supported*
H12b	*Help-seeking -> Self-Assessment*	*-.063*	*.045*	*1.402*	*.161*	*Not Supported*
H13b	Effort -> Self-Assessment	.257	.050	5.112***	.000	Supported
H14b	*Persistence -> Self-Assessment*	*.034*	*.055*	*.623*	*.534*	*Not Supported*
H15b	Deep Learning -> Self-Assessment	.175	.059	2.941***	.003	Supported
H16b	Surface Learning -> Self-Assessment	-.091	.049	1.863*	.063	Supported
H17b	*Trial and Error -> Self-Assessment*	*.047*	*.054*	*.877*	*.381*	*Not Supported*
H18b	*Interest -> Self-Assessment*	*.088*	*.060*	*1.449*	*.148*	*Not Supported*
H19b	Enjoyment -> Self-Assessment	.136	.059	2.287**	.022	Supported
H21b	Gratification -> Self-Assessment	.136	.058	2.332**	.020	Supported

Note: t-values > 1.65 (p<0.1); t-values > 1.96** (p<0.05); t-values > 2.57*** (p<0.01)*

7.6.1.3 Coefficient of Determination

To determine the research model's predictive accuracy, the coefficient of determination (R^2) value of the endogenous constructs was examined. The R^2 values are presented in Table 7.15. Three views exist on the range of acceptable R^2 values and are presented in Table 7.14. These views differ depending on the complexity of the model and the research discipline (Hair et al., 2014, p. 175).

Table 7.14: Comparison of acceptable R^2 values

Reference	Weak	Moderate	Substantial	Discipline
Chin (1998a)	.19	.33	.67	Information Systems (IS)
Cohen (1988)	.02	.13	.26	Behavioural Sciences
Hair et al. (2011)	.25	.50	.75	Marketing

Using Chin's (1998a) suggestion of the acceptable R^2 values, none of the constructs substantially predicted the endogenous constructs (dependent variable) *programming grade* and *self-assessment*. *Pre-programming self-efficacy* weakly predicted *post-programming self-efficacy* (.322).

Post-programming self-efficacy moderately predicted the variance in *deep learning* (.457), *enjoyment* (.390), and *persistence* (.400) while *post-programming self-efficacy* weakly predicted the variance in *gratification* (.221), *trial and error* (.266), *effort* (.313), and *interest* (.318). On the other hand, the R^2 values for *help-seeking* (.037) and *surface learning* (.074) were less than the acceptable threshold of .19 that was suggested by Chin.

In terms of predictability for programming performance, the engagement constructs moderately predicted the variance in the participants' *self-assessment* (.490) of their programming performance, and weakly predicted the variance in the participants' *programming grade* (.250).

7.6.1.4 Effect Size

To further determine the model's predictive accuracy, the effect size (f^2) of the lower-order engagement construct was examined. The f^2 *effect size* is used to evaluate "whether the omitted construct has a substantive impact on the endogenous constructs" (Hair et al., 2014, p. 177). A value of .02 may be interpreted as a small effect, .15 as a medium effect, and .35 as a large effect (Chin, 1998a; Cohen, 1988). Table 7.15 presents the effect sizes for the endogenous variables *programming grade* and *self-assessment*. Based on the outcome of the effect sizes, none of the lower-order engagement constructs that were omitted had a substantial impact on the endogenous construct *programming grade*, and *self-assessment*.

7.6.1.5 Blindfolding and Predictive Relevance

While the R^2 value measures the predictive accuracy, the Q^2 value (Fornell & Cha, 1994; Geisser, 1974; Stone, 1974) measures the predictive relevance of the model. Predictive relevance relates to the "accurate prediction of the data points of indicators in reflective measurement models of endogenous constructs and endogenous single-item constructs" (Hair et al., 2014, p. 178). Q^2 values should ideally be > 0 to have predictive relevance (Fornell & Cha, 1994).

The *blindfolding* technique is used to obtain the Q^2 value. At a specified point, the blindfolding technique omits a value in the indicators of the endogenous constructs and proceeds to evaluate the model based on the remaining values (Chin, 1998a; Henseler, Ringle, & Sinkovics, 2009). The omission distance (*D*) should be between 5 and 10, and the number of observations divided by the omission distance should not result in an integer value (Chin 1998a; Hair et al., 2014). Therefore, an omission distance of 7, and 433 observations was used to run the *blindfolding* procedure. Hair et al. (2014) recommended using the *cross-validated redundancy* approach to calculating the Q^2 value as the path model estimates from both the structural and the measurement model are used for predicting the relevance of the model (p. 183). Based on Table 7.16, all Q^2 values of the endogenous constructs were above 0. These values support the model's predictive relevance of the endogenous constructs.

The q^2 effect size was also calculated to measure the *relative* impact of the predictive relevance of the endogenous construct and is presented in Table 7.15. A q^2 effect size of .02 is small, .15 is medium while .35 is large (Henseler et al., 2009). None of the exogenous variables that were deleted demonstrated a substantial impact on the predictive relevance of the endogenous variables (*programming grade* and *self-assessment*). The effect size was not computed for the engagement constructs, and the post-programming self-efficacy construct as these constructs contained only one exogenous variable.

Table 7.15 Predictive accuracy and predictive relevance of the structural model

Construct	R^2	Programming Grade	Self-Assessment	Q^2	Programming Grade	Self-Assessment
		f^2	f^2		q^2	q^2
Post-programming self-efficacy	.322	*Not applicable*		.172	*Not applicable*	
Effort	.313	.038	.069	.149	.033	.032
Help Seeking	.037	.021	.006	.011	.015	.000
Persistence	.400	.000	.001	.203	-.003	-.001
Deep Learning	.457	.009	.025	.226	.008	.009
Surface Learning	.074	.051	.011	.034	.046	.003
Trial and Error	.266	.003	.002	.138	.000	.000
Interest	.318	.014	.006	.203	.006	.001
Enjoyment	.390	.033	.013	.238	.028	.006
Gratification	.221	.003	.020	.123	-.009	.009
Programming Grade	.250	*Not applicable*		.217	*Not applicable*	
Self-assessment	.490			.315		

7.6.1.6 Goodness-of-Fit

The goodness-of-fit of the research model was not estimated in this study since the existing goodness-of-fit index is inadequate for the validation of PLS-SEM models (Hair et al., 2014; Henseler & Sarstedt, 2013; Roberts & Grover, 2009). According to Hair et al. (2014, p. 101) the bootstrapping and blindfolding techniques (which are discussed in Section 7.6.1.2 and Section 7.6.1.5) are adequate in the estimation of the goodness-of-fit of the research model.

7.6.1.7 Final Structural Model (with lower-order engagement constructs)

Figure 7.4 presents the final structural model for the research. The grey arrows represent relationships that are not supported, and the black arrows present the relationships that are supported and the path coefficients of the relationships. The coefficient of determination (R^2) values of the endogenous constructs is also presented in each construct.

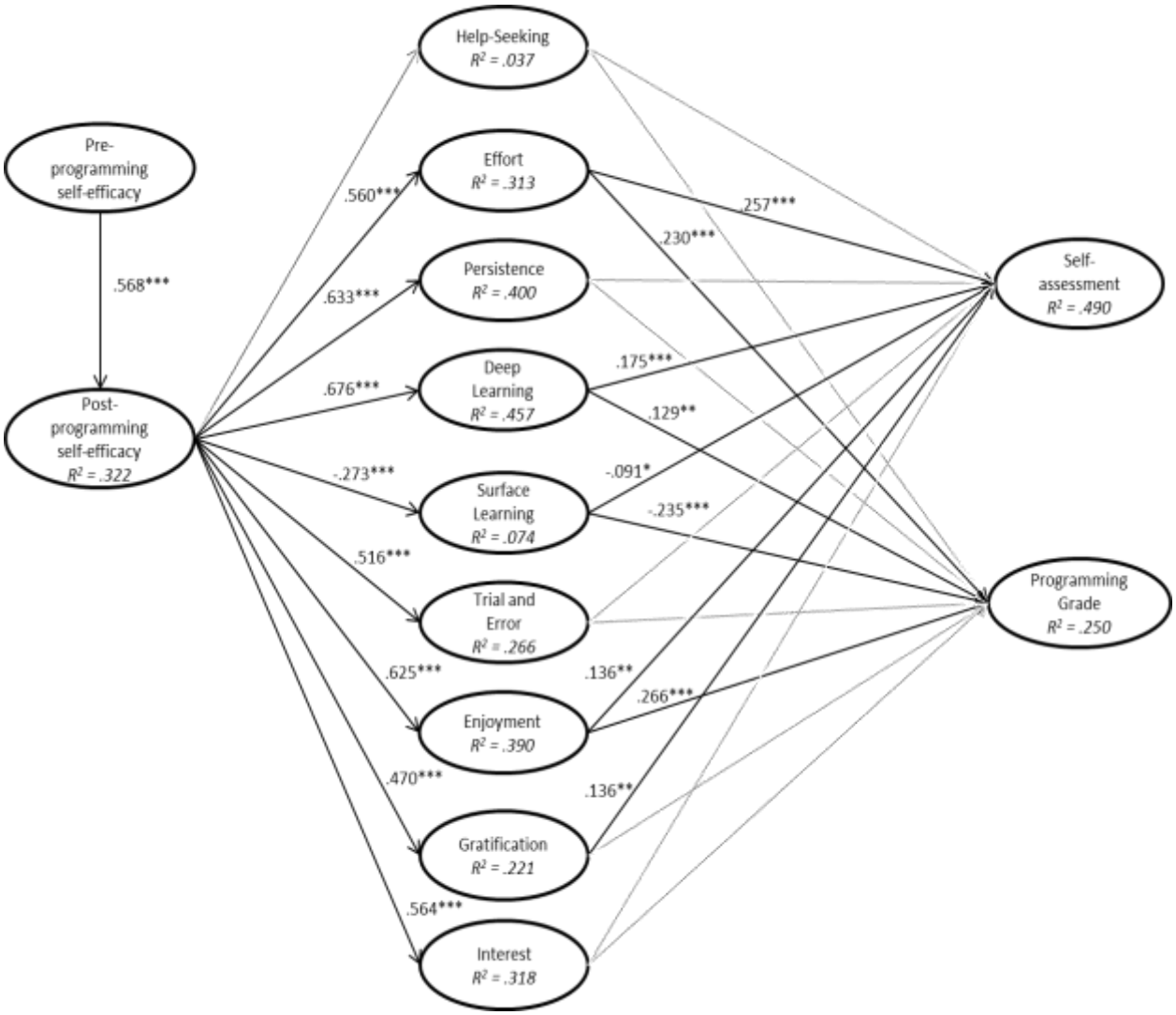


Figure 7.4: Final structural model with lower-order engagement constructs

7.7 Importance Performance-Matrix Analysis (IPMA)

IPMA is an analysis technique that compares the *importance* of the model to the *performance* of the model (Hair et. al, 2014). The *importance* is determined by the total effects of the structural model while the *performance* is determined by the average values of the latent variable (Hair et al., 2014). Hair et al. (2014) recommended the use of the IPMA to identify which construct(s) in the structural model are relatively *important* and/or have relatively higher *performance*.

The IPMA for *programming grade* and *self-assessment* were examined by estimating the performance and importance of the predecessor constructs. Table 7.16 displays the scores for the importance (total effects) and the performance (latent variable index values) of the direct and indirect predecessor of the *programming grade* and *self-assessment* constructs. The direct predecessors of the *programming grade* and *self-assessment* constructs are the lower-order engagement constructs while post-programming self-efficacy is the indirect predecessor of the *programming grade* and *self-assessment* constructs. Figures 7.5 and 7.6 show a graphical representation of the IPMA estimates for *programming grade* and *self-assessment*. The *x*-axis of the graph represents the importance (total effects) of the construct, and the *y*-axis represents the performance (latent variable index values) of the constructs. All direct and indirect predecessors of the endogenous constructs *programming grade* and *self-assessment* were included in the analysis.

Table 7.16: IPMA Scores

Construct	Programming Grade	Self-Assessment	Performance (Latent Variable Index Values)
	Importance (Total Effects)		
Post-programming Self-Efficacy	0.31	0.54	69.66
Pre-programming Self-Efficacy	0.18	0.31	57.54
Help-seeking	-0.15	-0.06	47.98
Effort	0.23	0.26	57.55
Persistence	-0.01	0.03	66.06
Deep Learning	0.13	0.18	66.49
Surface Learning	-0.24	-0.09	41.38
Trial and Error	-0.06	0.05	61.20
Enjoyment	0.27	0.14	70.58
Gratification	-0.07	0.14	74.22
Interest	-0.17	0.09	61.13

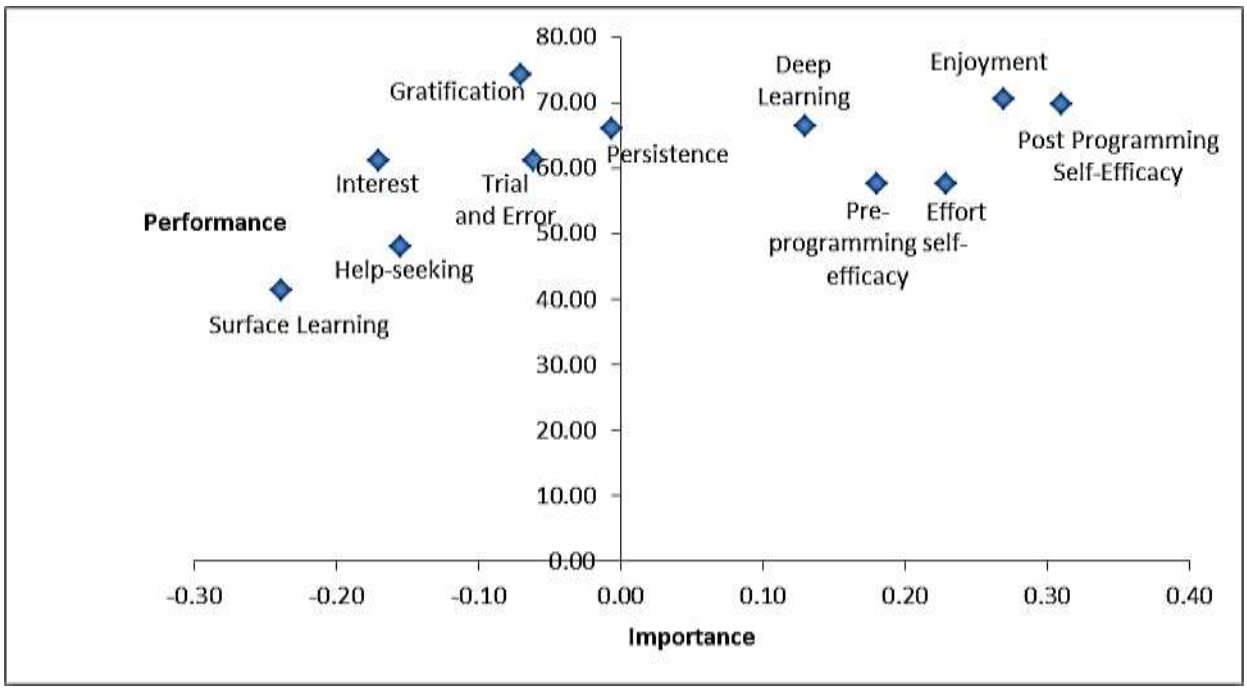


Figure 7.5: IPMA for Programming Grade

In Figure 7.5, the IPMA for *programming grade* showed that the direct predecessors *effort* and *enjoyment* ranked high in importance (total effects). *Enjoyment* was not only ranked high in importance but also ranked high in performance. This suggests that *enjoyment* was the most relevant lower-order engagement construct to *programming grade*. However, *surface learning* is the weakest predecessor in terms of performance and importance, and resulted in a negative performance to *programming grade*. *Post-programming self-efficacy*, which is an indirect predecessor of *programming grade* not only ranks high in performance but also in its importance to *programming grade* while *surface learning* was the least relevant lower-order engagement construct to *programming grade* since it was of least importance, and had the lowest performance.

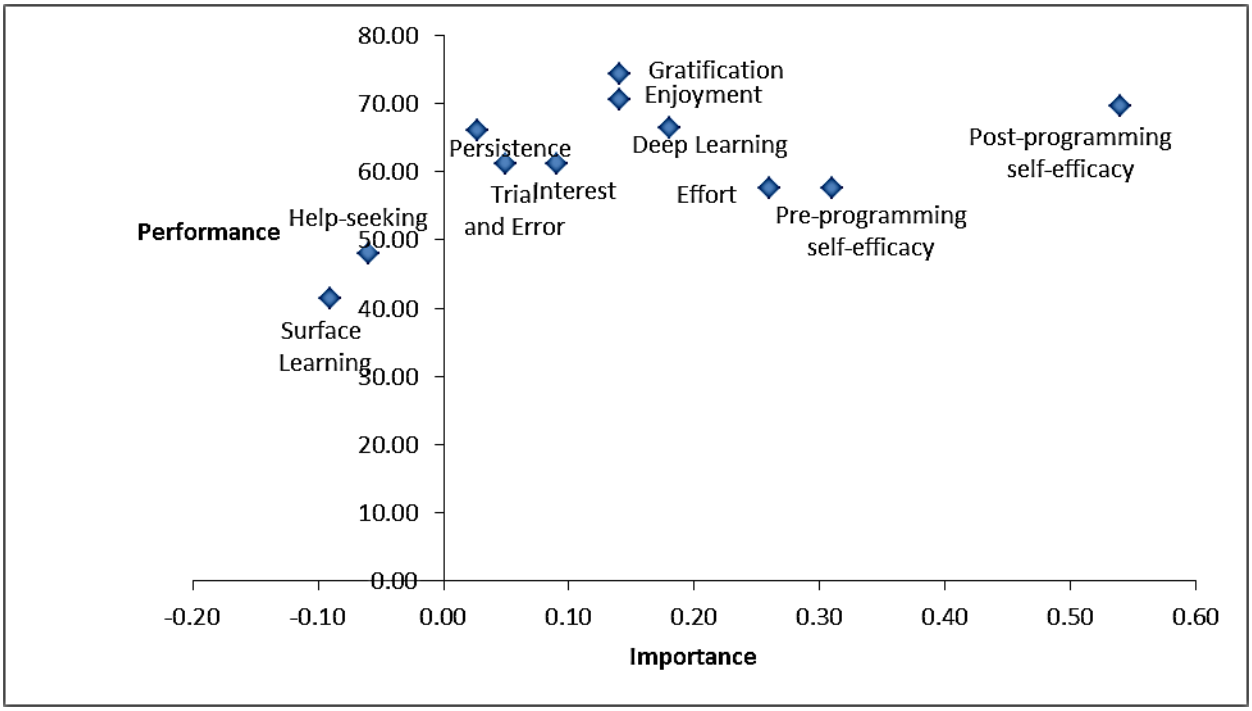


Figure 7.6: IPMA for Self-assessment

The IPMA for *self-assessment* was somewhat similar to the IPMA for *programming grade*. In Figure 7.6, the IPMA for *self-assessment* showed that the direct predecessors *effort*, *enjoyment*, and *gratification* ranked high in importance (total effects). *Gratification* and *enjoyment* were not only ranked high in importance but also ranked high in performance. This suggested that *gratification* and *enjoyment* were the most relevant lower-order engagement constructs to *self-assessment*. On the other hand, *surface learning* was the weakest predecessor in terms of performance and importance, and resulted in a negative performance to *self-assessment*. *Post-programming self-efficacy*, which is an indirect predecessor of *programming grade* not only ranked high in performance but also in its importance to *self-assessment* while *surface learning* was the least relevant lower-order engagement construct to *self-assessment*.

7.8 Confounding Variables

The confounding variables *programming experience* and *intelligence* were examined for its effect on the endogenous constructs, *programming grade* and *self-assessment*. The confounding variables were analysed for the significance of their path coefficients using the bootstrapping procedure, for its effect size, and were linked directly to the endogenous variables (*self-assessment* and *programming grade*).

To examine the effect of the confounding variables on the endogenous variables *self-assessment* and *programming grade*, three models were examined separately for their R^2 and path coefficients. The three models are:

Model 1: The hypothesised model *without* the confounding variables.

Model 2: The hypothesised model *with* the confounding variables.

Model 3: With the confounding variables only.

The outcomes of the analysis are presented in Table 7.17. Without the confounding variables (Model 1), the exogenous variables (lower-order engagement constructs) explained 25% *(R^2 = .250)* of the variance in *programming grade*, and 49% *(R^2 = .490)* of the variance in *self-assessment*. The following variances can be observed when each of the confounding variables was included into the hypothesised model (Model 2):

i. The variance in *programming grade* remained the same *(R^2 = .250)* when the *intelligence* confounding variable was included in the hypothesised model while the variance in *self-assessment* increased by .2% *(R^2 = .491)*. The path coefficients between *intelligence* and *programming grade (β = .008)*, and between *intelligence* and *self-assessment (β = .025)* were both less than <.1, which suggested non-significant relationships.

ii. The variance in *programming grade* and *self-assessment* remained the same *(R^2 = .250 and R^2 = .490 respectively)* when the *programming experience* confounding variable was included in the hypothesised model. The path coefficients between *programming experience* and *programming grade (β = .004)*, and between *programming experience* and *self-assessment (β = -.011)* were both less than < .1, which suggested non-significant relationships.

In the analysis of Model 3, the path coefficient for the relationship between *intelligence* and *programming grade* was < *.1* which suggested a non-significant relationship. The R^2 value of *.003* suggested that *intelligence* did not significantly predict *programming grade*. Next, the path coefficient of the relationship between *intelligence* and *self-assessment* was < *.2* which suggested a weak significance, and the R^2 value of *.010* suggested that *intelligence* did not significantly predict *self-assessment*.

Similarly, the path coefficient for the relationship between *programming experience* and *programming grade* was < *.1* which suggested a non-significant relationship. The R^2 value of *.008* suggested that *programming experience* did not significantly predict *programming grade*. Next, the path coefficient for the relationship between *programming experience* and *self-assessment* was < *.2* which suggested a non-significant relationship. Additionally, the R^2 value of *.023* suggested that *intelligence* did not significantly predict *self-assessment*.

Table 7.17: Comparison of the effect of confounding variables based on three types of models

Confounding Variable	Results		
	Model 1	**Model 2**	**Model 3**
Intelligence	Programming Grade *R^2: .250* Self-Assessment *R^2: .490*	Intelligence -> Programming Grade *Path coefficient: .008* *R^2: .250* Intelligence -> Self-Assessment *Path coefficient: .025* *R^2: .491*	Intelligence -> Programming Grade *Path coefficient: .052* *R^2: .003* Intelligence -> Self-Assessment *Path coefficient: .102* *R^2: .010*
Programming Experience		ProgExp -> Programming Grade *Path coefficient: .004* *R^2: .250* ProgExp -> Self-Assessment *Path coefficient: -.011* *R^2: .490*	ProgExp -> Programming Grade *Path coefficient: .091* *R^2: .008* ProgExp -> Self-Assessment *Path coefficient: .152* *R^2: .023*

Next, to confirm the significance of the path coefficients, the bootstrapping procedure was applied to Model 2 (the hypothesised model with the confounding variables). 5000 bootstrap samples were used to estimate the PLS path model (Hair et al., 2014). Table 7.18 shows the results of the bootstrapping analysis. The *t*-values for the relationships between *intelligence* and *programming grade*, *intelligence* and *self-assessment*, *programming experience* and *programming grade*, and *programming experience* and *self-assessment* showed that the relationships were not significant.

Table 7.18: Bootstrapping analysis for confounding variables

Confounding Variable	Endogenous Construct	
	Programming Grade	**Self-Assessment**
Intelligence	.008 [t = .186]	.025 [t = .688]
Programming Experience	.004 [t = .094]	-.011 [t = .262]

*Note: t-values > 1.65 * (p<0.1); t-values > 1.96 ** (p<0.05); t-values > 2.57 *** (p<0.01)*

7.9 Higher-order constructs

The engagement construct in the research model has thus far been examined at the lower-order level (or first-order construct). Hair et al. (2014) suggested that abstracting a model to a higher-order level may be required if supported by theory, and to achieve a more parsimonious model by reducing the number of relationships in the structural model. Since the engagement construct is a multi-dimensional construct consisting of *behavioural engagement*, *cognitive engagement*, and *emotional engagement* (Chapter 2, Section 2.3), the indicators of engagement were abstracted to the higher-order level and analysed.

The lower-order constructs *effort*, *persistence*, and *help-seeking* were abstracted into the higher-order construct *behavioural engagement* illustrated in Figure 7.7. The lower-order constructs *deep learning*, *surface learning*, and *trial and error* were abstracted into the higher-order construct *cognitive engagement* illustrated in Figure 7.8 while the lower-order constructs *interest*, *enjoyment*, and *gratification* was abstracted into the higher-order construct *emotional engagement* illustrated in Figure 7.9.

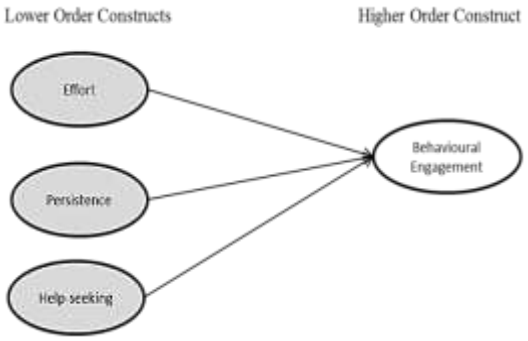


Figure 7.7 Lower-order constructs for Behavioural Engagement

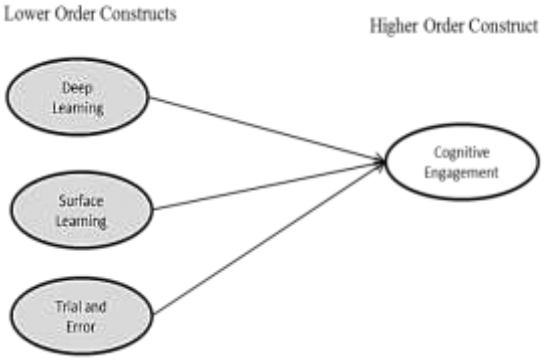


Figure 7.8 Lower-order constructs for Cognitive Engagement

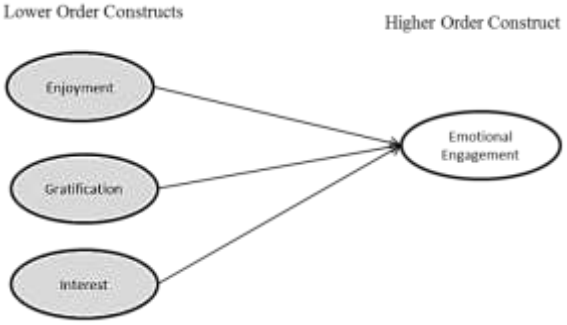


Figure 7.9 Lower-order constructs for Emotional Engagement

The higher-order constructs were analysed using the reflective-formative model since in Chapter 3, Section 3.4, we established that the measurement model for the lower-order constructs should be measured reflectively, and that the relationship between the lower-order constructs and the higher-order constructs should be measured formatively.

7.9.1 Measurement Model

The repeated indicators approach was used as the initial measurement model of the higher-order constructs (Hair et al., 2014; Lohmöller, 1989). In this approach, the indicators of the lower-order constructs are repeated as indicators of the higher-order constructs.

Since the repeated indicator approach resulted in the higher-order construct having a variance that was explained by all of its lower-order constructs ($R^2 \approx 1$), a two-stage approach was used to set up the reflective-formative model (Henseler & Chin, 2010). In the first stage, the items that measure the lower order constructs were repeated in the higher-order construct in order to obtain the latent variable scores and are illustrated in Figure 7.10. The reflective indicators of the lower-order construct *interest*, *enjoyment*, and *gratification* are repeated as reflective indicators of the higher-order construct *emotional engagement*. In the second stage, the lower-order constructs then become manifest variables in the measurement model of the higher-order construct and are illustrated in Figure 7.11.

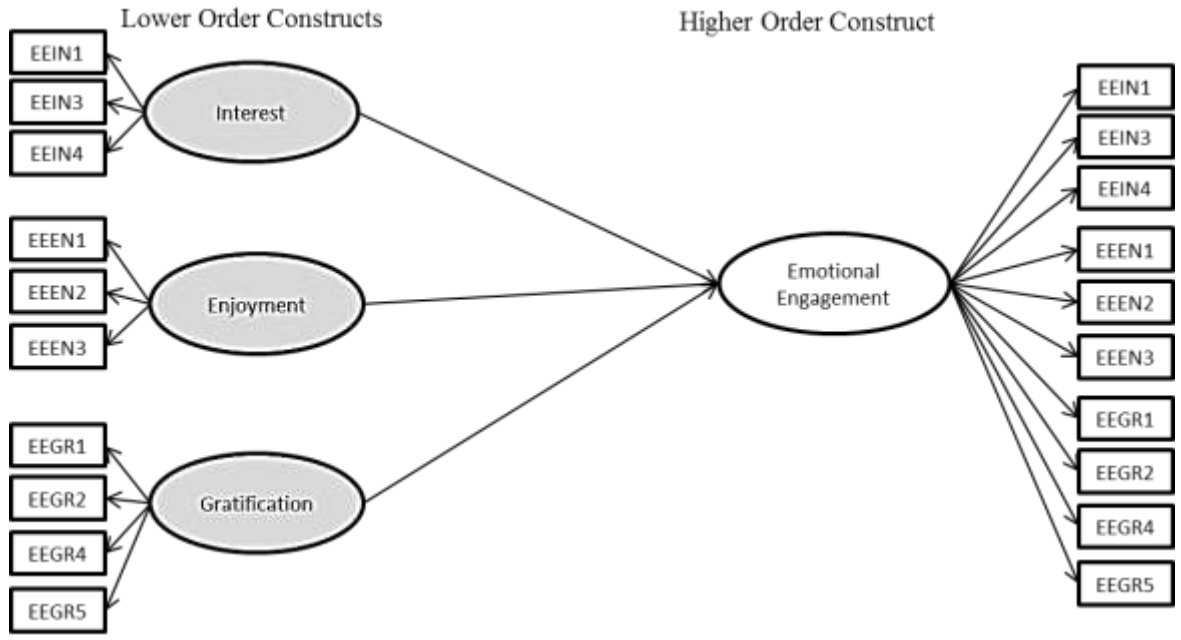


Figure 7.10: Stage 1 – Example of Repeated Indicator Approach

Figure 7.11: Stage 2 – Higher-order measurement model for engagement constructs

Since the reliability and validity of the measurement model of the lower-order constructs were examined in the earlier section (Section 7.5.2, Section 7.5.3, Section 7.5.4), the focus of this section was to examine the validity of the measurement model for the higher-order constructs. The tests to determine the validity of a formative measurement model differ from those for a reflective measurement model (Chin, 1998a). Diamantopoulos (2006) explained that formative indicators are error free, thereby eliminating the need to establish the consistency reliability (CR) of the measurement model. Instead, Hair et al. (2014) suggested that the higher-order measurement model should be assessed for content validity of the formatively measured construct, convergent validity, collinearity issues, and to finally assess the significance and relevance of the formative indicators (p. 121). As the content validity of the model had been established during the development of the survey questionnaires (Chapter 6), this section focuses on assessing the convergent validity, collinearity issues, and finally assesses the significance and relevance of the formative indicators of the higher-order measurement model.

7.9.1.1 Convergent Validity

Chin (1998a) proposed the use of redundancy analysis to measure the convergent validity of a formatively measured construct. Using the redundancy analysis approach, a relationship was established between the same construct, with one construct being measured formatively, and the other measured reflectively (as illustrated in Figures 7.12, 7.13, 7.14). The formatively measured construct was modelled as the predictor variable, and the reflectively measured construct was modelled as the endogenous construct.

During the development of the survey questionnaire, reflectively measured redundant items were also included in the questionnaire in order to measure the higher-order *behavioural engagement*, *cognitive engagement*, and *emotional engagement* constructs (Chapter 6). For example, in Figure 7.12, BEGI1 and BEGI2 are the redundant items for the reflectively measured *behavioural engagement* construct.

Chin (1998a) then recommends that the magnitude of the relationship between the formatively and reflectively measured constructs should be $\geq .8$ while the R^2 should be $\geq .64$. All three higher-order engagement constructs satisfied the conditions recommended by Chin and are presented in Figures 7.12, 7.13, and 7.14. The magnitude of the *behavioural engagement* construct was *.860* and $R^2 = .740$ while the magnitude of the *cognitive engagement* construct was *.842* and $R^2 = .710$, and finally, the magnitude of the *emotional engagement* construct was *.861* and $R^2 = .741$.

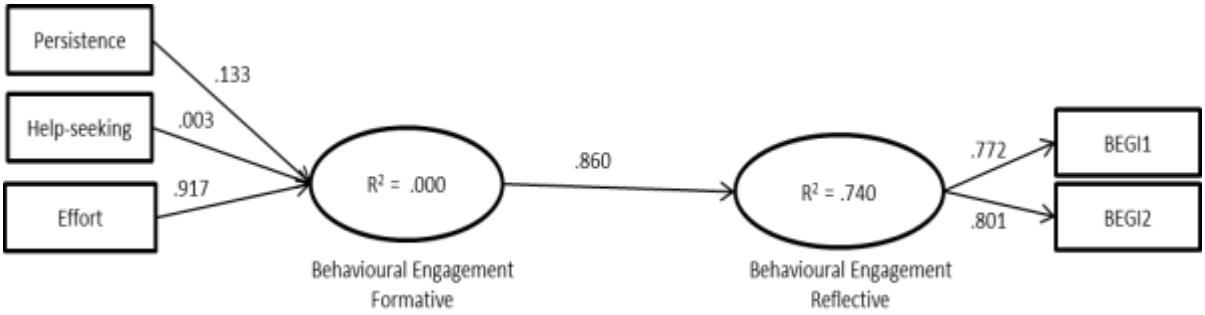


Figure 7.12: Convergent Validity for Behavioural Engagement

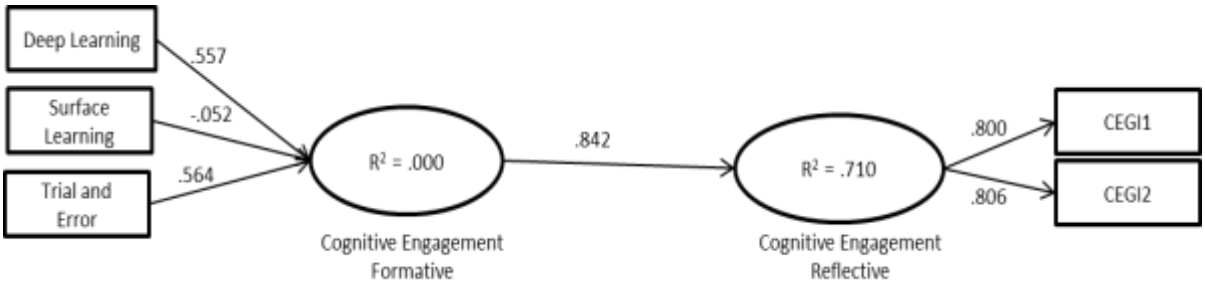


Figure 7.13: Convergent Validity for Cognitive Engagement

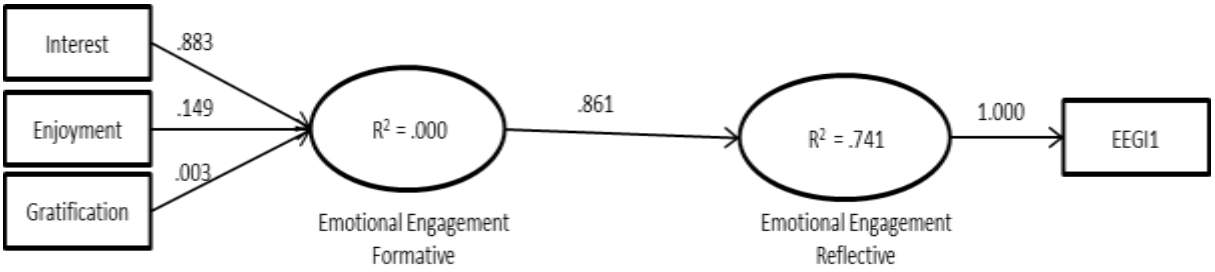


Figure 7.14: Convergent Validity for Emotional Engagement

7.9.1.2 Collinearity Issues

The measurement model was examined for collinearity issues as high levels of collinearity could affect the statistical significance of the weights of the formative constructs (Hair et al., 2014). The results of the analysis are presented in Table 7.19. A multiple regression analysis was performed on all the indicators of the higher-order engagement constructs using SPSS (Version 20.0). The highest VIF value among the indicators was *2.550* (*Enjoyment*), which was well below the threshold VIF value of *5* (Hair et al., 2011). Additionally, the lowest tolerance level among the indicators was *.392* (*Enjoyment*) which was well above the proposed tolerance value of *.2* (Hair et al., 2011). These results confirm that there were no collinearity issues between each set of indicators of the higher-order model.

Table 7.19: Collinearity statistics for the higher-order measurement model

Indicators	Behavioural Engagement		Cognitive Engagement		Emotional Engagement	
	Tolerance	**VIF**	**Tolerance**	**VIF**	**Tolerance**	**VIF**
Effort	.492	2.034				
Persistence	.408	2.450				
Help-seeking	.754	1.326				
Deep Learning			.426	2.349		
Surface Learning			.746	1.341		
Trial and Error			.482	2.073		
Interest					.443	2.260
Enjoyment					.392	2.550
Gratification					.527	1.896

7.9.1.3 Significance and relevance of the formative indicators

The final assessment of the formative measurement model was in the estimation of the significance of its weights (Hair et al., 2014). The outer weights of the indicators were used to examine the relative importance of each indicator to the construct (Hair et al., 2014, p. 127). A bootstrapping procedure was carried out to test the significance of the outer weights, and if the formative indicators significantly contributed to the higher-order engagement construct. 5000 bootstrap samples were used to estimate the weights of the formative indicators (Hair et al., 2014). Table 7.20 presents the outcome of the bootstrapping analysis. In summary, the *t*-value of the outer weights confirmed that, with the exception of *interest*, all other formative indicators were significant at a 1% probability of error while *interest* was significant at a 5% probability of error. The significance of the outer weights confirmed that all the formative indicators were relatively important to their respective constructs.

Table 7.20: Significance of the formative indicators in the measurement model

Construct	Formative Indicator	Outer Weights	Standard Error	t-value	p- Values
Behavioural Engagement	Effort	.542	.056	9.631***	.000
	Persistence	.540	.058	9.280***	.000
	Help-seeking	-.364	.053	6.881***	.000
Cognitive Engagement	Deep Learning	.699	.059	11.753***	.000
	Surface Learning	-.336	.051	6.555***	.000
	Trial and Error	.311	.068	4.560***	.000
Emotional Engagement	Enjoyment	.640	.081	7.944***	.000
	Gratification	.291	.065	4.485***	.000
	Interest	.208	.087	2.384**	.017

Note: t-values > 1.65 (p<0.1); t-values > 1.96** (p<0.05); t-values > 2.57*** (p<0.01)*

7.9.1.4 Final Higher-order Measurement Model

Figure 7.15 presents the final higher-order measurement model for this study after satisfying the conditions for convergent validity.

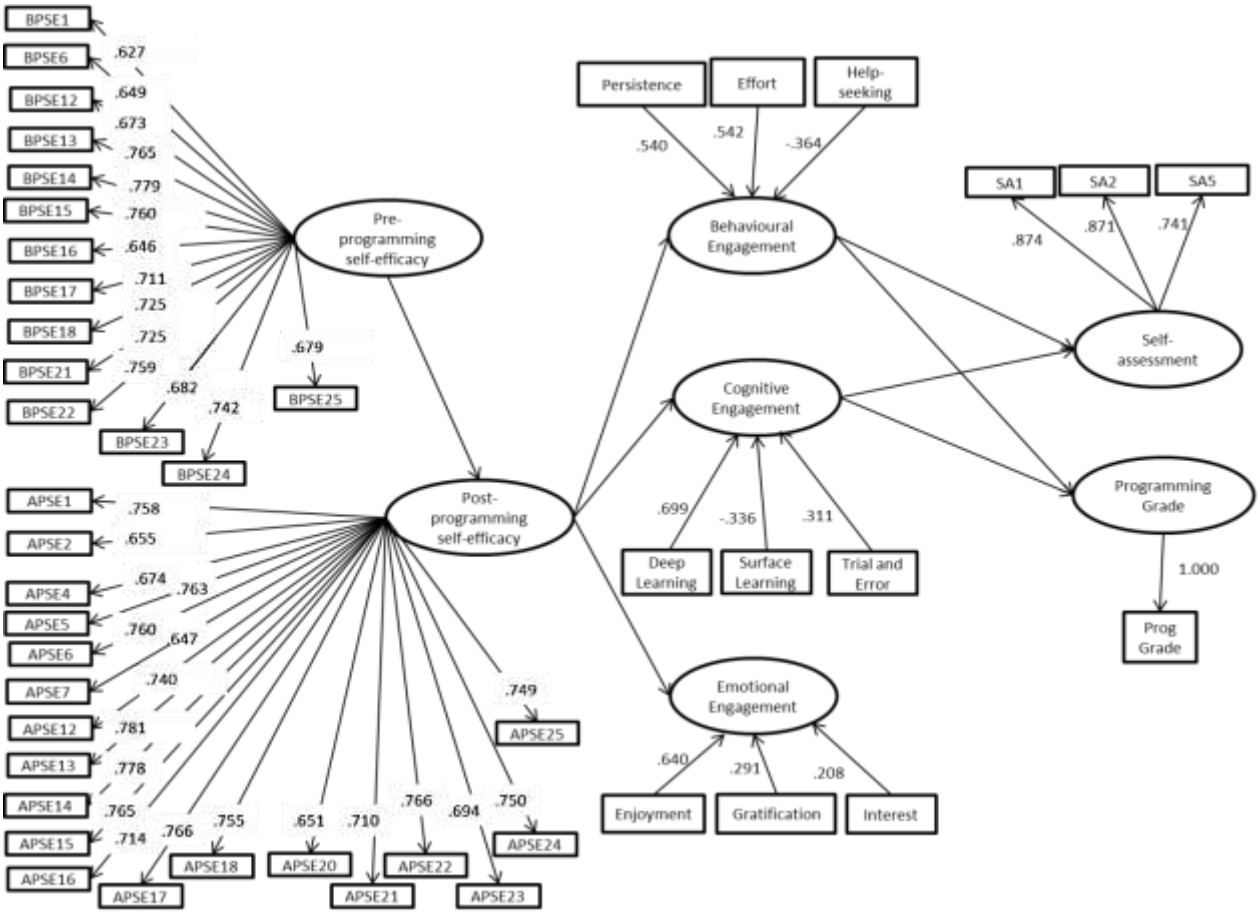


Figure 7.15: Higher-order measurement model

7.9.2 Structural Model (with higher-order engagement constructs)

Section 7.9.1 assessed and confirmed the validity and reliability of the higher-order measurement model. To assess the structural model of the higher-order construct, the five steps that were applied to analyse the structural model with lower-order engagement constructs in Section 7.6 was repeated for the higher-order constructs.

7.9.3 Collinearity Issues

A multiple regression analysis was performed on all the predictor variables of the structural model using SPSS (Version 20.0). The results of the analysis are presented in Table 7.21. All predictor variables had a VIF value below the threshold VIF value of *5* (Hair et al., 2011). Additionally, the tolerance level of the predictor variables was above the proposed tolerance value of *.2* (Hair et al., 2011). These results confirm that there were no collinearity issues between each set of predictor variables in the structural model with higher-order engagement constructs.

Table 7.21: Collinearity Statistics for the higher-order structural model

Predictor Variables	Programming Grade		Self-Assessment	
	Tolerance	VIF	Tolerance	VIF
Behavioural Engagement	.450	2.224	.450	2.224
Cognitive Engagement	.431	2.318	.431	2.318
Emotional Engagement	.492	2.034	.492	2.034

7.9.4 Hypothesis Testing using Bootstrapping

The bootstrapping procedure was used to determine the significance of the path coefficients of the structural model with higher-order engagement constructs. 5000 bootstrap samples were used to estimate the PLS path model (Hair et al., 2014). Table 7.22 presents the outcome of the hypothesis testing using the bootstrapping procedure. The "no sign change" option was used to run the bootstrap procedure. The path coefficients are interpreted in the same manner as discussed in Section 7.6.1.2.

Relationship between post programming self-efficacy and higher-order engagement constructs

There was a strong positive relationship between *post-programming self-efficacy* and the higher-order engagement constructs and was significant at a 1% probability of error. Thus, hypotheses HPSEB *(β=.715; t=27.336),* HPSEC *(β=.725; t=28.193),* and HPSEE *(β=.654; t=23.862)* were supported.

Relationship between higher-order engagement constructs and programming grade

There was a strong positive relationship between *behavioural engagement* and *programming grade*, and between *cognitive engagement* and *programming grade* and the relationships were significant at a 1% probability of error. Thus, hypothesis HBPG *(β=.181; t=2.624)* and hypothesis HCPG *(β=.210; t=2.801)* were supported.

The relationship between *emotional engagement* and *programming grade* was not significant. Thus, hypothesis HEPG *(β=.019; t=.293)* was not supported.

Relationship between higher-order engagement constructs and self-assessment

There was a strong positive relationship between the higher-order *engagement* constructs and the *self-assessment* and the relationships were significant at a 1% probability of error. Thus, hypotheses HBSA (*β=.268; t=5.180)*, HCSA *(β=.217; t=4.114)*, and HESA *(β=.296; t=6.264)* were supported.

7.9.5 Coefficient of Determination

To determine the predictive accuracy of the structural model with higher-order engagement constructs, the coefficient of determination (R^2) value of the endogenous constructs was examined. The R^2 values are presented in Table 7.23.

Using Chin's suggestion of the acceptable R^2 values (Section 7.6.1.3) none of the constructs substantially predicted the endogenous constructs. However, *post-programming self-efficacy* moderately predicted the variance in *behavioural engagement* ($R^2=.511$), *cognitive engagement* ($R^2=.525$), and *emotional engagement* ($R^2=.428$)

As for predictability for programming performance, the higher-order engagement constructs moderately predicted the variance in the participants' *self-assessment* ($R^2=.476$) of their programming performance. However, the higher-order engagement constructs had little predictive accuracy on the novice programmer's *programming grade* ($R^2=.140$).

7.9.6 Effect size

To further determine the model's predictive accuracy, the effect size (f^2) of the higher-order engagement constructs was examined. Cohen's (1988) guidelines for interpreting the effect sizes were applied (Section 7.6.1.4). Table 7.23 presents the effect sizes for the endogenous variables *programming grade* and *self-assessment*. Based on the outcome of the effect sizes, none of the higher-order engagement constructs that were omitted had a substantive impact on the endogenous construct *programming grade*, and *self-assessment*.

Table 7.22: Hypothesis testing using Bootstrapping (with higher-order engagement constructs)

Hypothesis	**Relationship**	**Path Coefficient**	**Standard Error**	**t-value**	***p*-value**	**Decision**
HPSEB	Post-programming self-efficacy -> Behavioural Engagement	.715	.026	27.336***	.000	Supported
HPSEC	Post-programming self-efficacy -> Cognitive Engagement	.725	.026	28.193***	.000	Supported
HPSEE	Post-programming self-efficacy -> Emotional Engagement	.654	.027	23.862***	.000	Supported
HBPG	Behavioural Engagement -> Programming Grade	.181	.069	2.624***	.009	Not Supported
HCPG	Cognitive Engagement -> Programming Grade	.210	.075	2.801***	.005	Supported
HEPG	*Emotional Engagement -> Programming Grade*	*.019*	*.065*	*.293*	*.769*	*Not Supported*
HBSA	Behavioural Engagement -> Self-Assessment	.268	.052	5.180***	.000	Supported
HCSA	Cognitive Engagement -> Self-Assessment	.217	.053	4.114***	.000	Supported
HESA	Emotional Engagement -> Self-Assessment	.296	.047	6.246***	.000	Supported

Note: t-values > 1.65 (p<0.1); t-values > 1.96** (p<0.05); t-values > 2.57*** (p<0.01)*

7.9.7 Blindfolding and Predictive Relevance

The *blindfolding* technique was used to obtain the Q^2 value (Section 7.6.1.5). An omission distance of 7 and 433 observations was used to run the *blindfolding* procedure. The *cross-validated redundancy* approach was used to calculate the Q^2 value. In Table 7.23, all the Q^2 values of the endogenous constructs were above *0*. These values supported the model's predictive relevance of the endogenous constructs.

The q^2 effect sizes were calculated to measure the *relative* impact of the predictive relevance of the endogenous construct and are presented in Table 7.23. None of the exogenous variables (higher-order engagement constructs) that were deleted demonstrated an impact on the predictive relevance of the endogenous variable *programming grade*. However, all three of the engagement constructs had a small impact on the predictive relevance of the endogenous variable *self-assessment* when each of the higher-order engagement constructs was deleted from the structural model.

Table 7.23: Predictive accuracy and predictive relevance of the structural model (with higher-order engagement constructs)

Construct	R^2	**Programming Grade**	**Self-Assessment**	Q^2	**Programming Grade**	**Self-Assessment**
		f^2	f^2		q^2	q^2
Post-programming self-efficacy	.322	*Not applicable*		.321	*Not applicable*	
Behavioural Engagement	.511	.017	.062	.244	.016	.056
Cognitive Engagement	.525	.022	.039	.261	.016	.034
Emotional Engagement	.428	.000	.082	.305	-.003	.077
Programming Grade	.140	*Not applicable*		.129	*Not applicable*	
Self-assessment	.476			.467		

7.9.8 Final Structural Model (with higher-order engagement constructs)

Figure 7.16 presents the final structural model for the *behavioural engagement*, *cognitive engagement*, and the *emotional engagement* constructs.

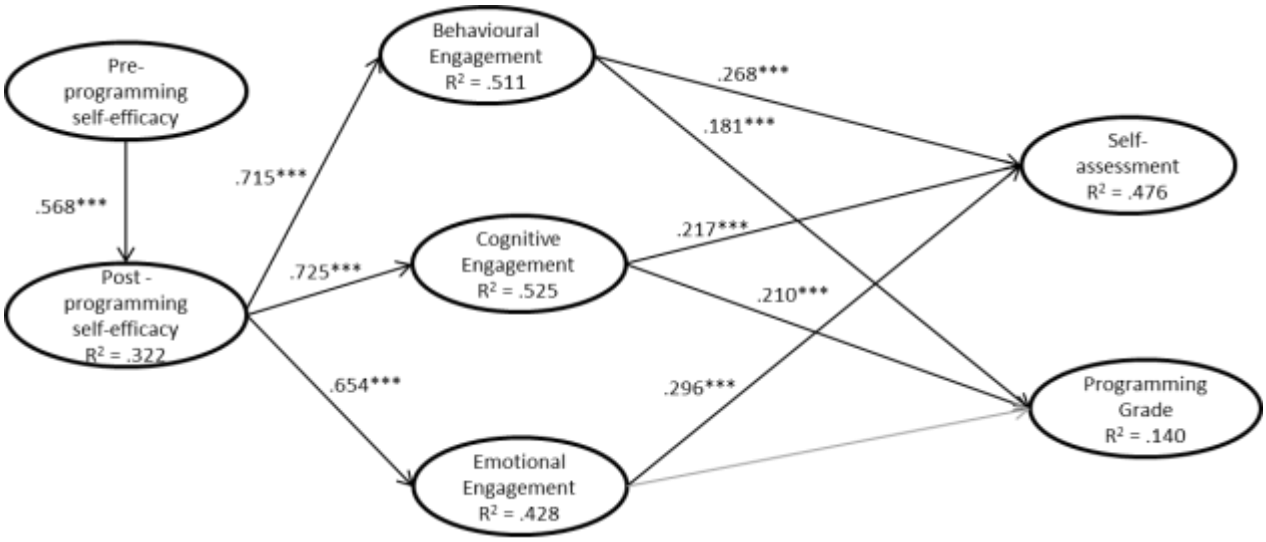


Figure 7.16: Structural model for higher-order engagement construct

7.10 Multiple Mediation

Hair et al. (2014) explained that a mediating effect occurs when "a third construct intervenes between two other related constructs" (p. 35). Mediating effects are normally made when there is theoretical evidence (Baron & Kenny, 1986; Hair et al., 2014; Preacher & Hayes, 2008). With the exception of the assertion by Appleton et al. (2006), no further literature on learning theory could be found which argued that engagement created a mediating effect between self-efficacy and performance. Instead, the literature on self-efficacy suggested that self-efficacy is a predictor of behaviour rather than performance (Bandura, 1977; Bandura et al., 1977; Zimmerman, 1995) (Chapter 2, Section 2.1.4.2). Due to the lack of evidence in the literature on learning theory and self-efficacy, the structural model (with higher-order engagement constructs) was nevertheless tested for the possibility that the multi-dimensional engagement construct might mediate the effect of programming self-efficacy on the programming performance of the novice programmer.

The first consideration for mediation was to determine if the model required a simple (but separate) or a multiple mediation test. As engagement is a multi-dimensional construct consisting of behavioural engagement, cognitive engagement, and emotional engagement, a multiple mediator analysis test was conducted on the data. Preacher and Hayes (2008, p. 881) argued that multiple mediation is preferred over simple mediation due to the following reasons:

a. The test for the indirect effect of *post-programming self-efficacy* on *programming grade* and *self-assessment* is similar to conducting a multiple regression on several predictors and attempts to determine if an overall effect exists.
b. The effect of each mediating variable may be determined although the other mediating variables will also be present in the model.
c. The parameter bias is reduced since all the mediating variables are considered in the test, and none will be dropped from the analysis.
d. The magnitude of the indirect effect of each mediator may be determined and compared.

Appendix G details the steps that were taken to perform the multiple mediation and the outcomes of the analysis. In summary, the outcome of the mediation analysis suggested that the *behavioural engagement* and *emotional engagement* constructs mediated the relationship between *post-programming self-efficacy* and *self-assessment*, and the higher-order engagement constructs cumulatively mediated the relationship between *post-programming self-efficacy* and *self-assessment*.

7.11 Multi-group Analysis

To further understand the effect of self-efficacy on the engagement behaviour of the novice programmers, and the effect of their engagement on their programming performance, the dataset was divided into meaningful groups so that they may be compared. Hair et al. (2014) referred to the significant differences in the relationships between groups as heterogeneity and argued that comparing between groups is important to understand the different outcomes (p. 244). The Partial Least Squares – Multi-Group Analysis (PLS-MGA) test in SmartPLS 3 was used to examine the groups.

The comparisons were made by country, programming grade, intelligence, programming self-efficacy, and gender. Like the analysis of any PLS path model, the sample size of each group should adhere to the minimum sample size requirement of the "maximum number of arrows pointing to a latent variable in the structural model multiple by 10" (Hair et al., 2014, p. 250) In this study, the maximum number of arrows pointing to any one latent variable in the structural model was 9 (for example, there are 9 arrows pointing to the latent variable programming grade). Therefore, the minimum sample size for each group should ideally be (9 arrows * 10) = 90.

Prior to running the PLS-MGA test, the convergent and discriminant validities were determined for all groups, and the bootstrapping analysis was performed. All groups meet the validity criteria for convergent and discriminant validity. The bootstrap analysis was performed using 5,000 samples as suggested by Hair et al. (2014), and the number of cases was equal to the number of samples for each group (represented by *n* in the tables in each sub-section).

The PLS-MGA test provides results of the bootstrap analysis and the results of the significance of the relationships between groups. The nonparametric confidence interval approach proposed by Sarstedt, Henseler, and Ringle (2011) was used to analyse the significance of the relationships between groups. There are three approaches to analysing multi-groups. They are the parametric approach (Keil et al., 2000), the permutation-based approach (Chin, 2003), and Henseler's PLS Multigroup Analysis (Henseler, 2007). According to Sarstedt et al. (2011), these three approaches allow only a one-sided hypothesis test compared to the nonparametric confidence set approach.

Using the nonparametric confidence set approach, the significance of the relationships between groups was determined by comparing the path coefficients of a relationship in for example group A to the confidence intervals of group B. If the path coefficient of the relationship in group A fall within the confidence intervals of group B or if the path coefficient of the relationship in group B fall within the confidence intervals of group A, then there are no significant differences between group A and group B at a significance level α (Sarstedt et al., 2011). However, if the path estimate does not fall into the confidence intervals of the other group, then the path coefficient of the group can be assumed to be significantly different (Sarstedt et al., 2011).

7.11.1 Comparison by Country

To compare by country, the data was grouped based on the countries where the data was collected. Table 7.24 presents the path coefficients and the significance of the relationships between the Malaysia and the New Zealand group by using the bias-corrected 95% confidence intervals approach (Shi, 1992).

Relationship between post-programming self-efficacy and lower-order engagement constructs

The relationship between *post-programming self-efficacy* and *help-seeking* had a positive effect on the Malaysia (β =*.317*) group but a negative effect on the New Zealand (β = -*.276*) group. The relationship between *post-programming self-efficacy* and *surface learning* was stronger on the New Zealand (β = -*.341*) group compared to the Malaysia (β = -*.050*) group.

Relationship between lower-order engagement constructs and programming grade

The relationship between *effort* and *programming grade* had a negative effect on the Malaysia (β = -*.017*) group, but a positive effect on the New Zealand (β = *.413*) group. The relationship between *interest* and *programming grade* had a negative effect on the Malaysia (β = -*.208*) group, but a positive effect on the New Zealand (β = *.015*) group.

Relationship between lower-order engagement constructs and self-assessment

The relationship between *surface learning* and *self-assessment* was stronger on the New Zealand (β = *-.188*) group compared to the Malaysia (β = *.000*) group.

Table 7.24: Multi-group comparison by Country

Relationships	Path Coefficients		Confidence Intervals				Sig.
	Malaysia (*n = 219*)	New Zealand (*n = 214*)	Malaysia		New Zealand		
			CI (Low)	*CI (High)*	*CI (Low)*	*CI (High)*	
Post-programming self-efficacy -> Help-seeking	0.317***	-0.276***	0.249	0.485	-0.441	-0.179	Sig.
Post-programming self-efficacy -> Surface Learning	-0.050	-0.341***	-0.244	0.174	-0.512	-0.239	Sig.
Effort -> Programming Grade	-0.017	0.413***	-0.239	0.147	0.261	0.559	Sig.
Surface Learning -> Self-Assessment	0.000	-0.188**	-0.143	0.146	-0.340	-0.049	Sig.
Interest -> Programming Grade	-0.208**	0.015	-0.396	-0.021	-0.186	0.183	Sig.

Sig. denotes a significance at (p <= 0.05); Nsig. denotes a non-significant relationship
Note: t-values > 1.65 (p<0.1); t-values > 1.96** (p<0.05); t-values > 2.57*** (p<0.01)*

7.11.2 Comparison by Intelligence

The multi-group comparisons for *intelligence* were divided into 3 groups (which are average, good and excellent). Since intelligence had more than 2 groups, using the traditional pairwise method to compare between groups may result in an error rate that is beyond the acceptable threshold of the Type-I error level (Sarstedt et al., 2011). Therefore, the omnibus test of group differences (OTG) approach was applied (Sarstedt et al., 2011) to control the error rates. Sarstedt et al. (2011) argued that the OTG is preferred over the Bonferroni correction approach or the ANOVA approach as "the OTG strategy is an optimal test which is able to maintain the familywise error rate, deliver an acceptable level of statistical power, and does not rely on distributional assumptions" (p. 206).

To meet the minimum sample size requirement, the grading scheme for intelligence was re-coded from 5 categories to 3 categories as presented in Table 7.25. Table 7.26 then presents the path coefficients and the significance of the relationships between the three *intelligence* groups by using the bias-corrected 95% confidence intervals approach (Shi, 1992).

Table 7.25: Categories for Intelligence PLS-MGA

Initial Categories	Marks	New Categories for PLS-MGA
A	80% - 100%	Excellent
B	65% - 79%	Good
C	50% - 64%	Average
D	30% – 49%	
E	<30%	

Relationship between pre-programming self-efficacy and post-programming self-efficacy

The relationship between *pre-programming self-efficacy* and *post-programming self-efficacy* was stronger in the average scores (*β = .711*) group compared to the good scores (*β = .501*) group. The same can be observed with the average scores (*β =.711*) and excellent scores (β = .482) groups whereby the effect of *pre-programming self-efficacy* on *post-programming self-efficacy* was stronger in the average intelligence score group.

Relationship between post-programming self-efficacy and lower-order engagement constructs

The relationship between *post-programming self-efficacy* and *enjoyment* had a stronger positive effect on the average scores (*β = .686*) group compared to the good scores (*β = .586*) group. The relationship between *post-programming self-efficacy* and *persistence* had a stronger positive effect on the average scores (*β = .702*) group compared to the good scores (*β = .539*) group. The relationship between *post-programming self-efficacy* and *trial and error* had a stronger positive effect on the average scores (*β = .601*) group compared to the good scores (*β = .373*) group. The same can be observed between the good scores (*β =.373*) and excellent scores (*β = .639*) groups. There is a stronger positive effect in the excellent intelligence score group compared to the good score group.

Relationship between lower-order engagement constructs and programming grade

The relationship between *interest* and *programming grade* had a stronger negative effect on the good score (*β = -.271*) group compared to the average score (*β = -.039*) group.

Relationship between lower-order engagement constructs and self-assessment

The relationship between *trial and error* and *self-assessment* had a negative effect on the average score (*β =- .127*) group compared to a positive effect on the excellent score (*β = .156*) group.

Table 7.26: Multi-group comparison by Intelligence

Relationships	Path Coefficient			Confidence Intervals						Comparison	Sig.
	Average (*n=126*)	Good (*n=203*)	Excellent (*n=104*)	Average		Good		Excellent			
				CI (Low)	*CI (High)*	*CI (Low)*	*CI (High)*	*CI (Low)*	*CI (High)*		
Pre-programming self-efficacy -> Post-programming self-efficacy	0.711***	0.501***	0.482***	0.632	0.789	0.379	0.626	0.352	0.653	Average vs. Good Average vs. Excellent	Sig. Sig.
Post-programming self-efficacy -> Enjoyment	0.686***	0.586***	0.622***	0.621	0.778	0.519	0.673	0.517	0.749	Average vs. Good	Sig.
Post-programming self-efficacy -> Persistence	0.702***	0.539***	0.676***	0.659	0.791	0.441	0.665	0.550	0.811	Average vs. Good	Sig.
Post-programming self-efficacy -> Trial and Error	0.601***	0.373***	0.639***	0.530	0.722	0.283	0.525	0.478	0.778	Average vs. Good Good vs. Excellent	Sig. Sig.
Interest -> Programming Grade	-0.039	-0.271***	-0.129	-0.227	0.236	-0.458	-0.068	-0.436	0.173	Average vs. Good	Sig.
Trial and Error -> Self-Assessment	-0.127	0.079	0.156	-0.281	0.072	-0.031	0.223	-0.082	0.404	Average vs. Excellent	Sig.

Sig. denotes a significance at (p <= 0.05); Nsig. denotes a non-significant relationship
Note: t-values > 1.65 (p<0.1); t-values > 1.96** (p<0.05); t-values > 2.57*** (p<0.01)*

7.11.3 Comparison by Programming Grade

Similar to the comparison by intelligence, the multi-group comparisons for *programming grade* consisted of 3 groups (which are average, good and excellent). The omnibus test of group differences (OTG) approach was applied (Sarstedt et al., 2011) to control the error rates.

To meet the minimum sample size requirement, the *programming grade* scores were re-coded from 5 categories to 3 categories and are presented in Table 7.27. Table 7.28 presents the path coefficients and the significance of the relationships between the three *programming grade* groups by using the bias-corrected 95% confidence intervals approach (Shi, 1992).

Table 7.27: Categories for Programming Grade PLS-MGA

Initial Categories	Marks	New Categories for PLS-MGA
A	80% - 100%	Excellent
B	65% - 79%	Good
C	50% - 64%	Average
D	40% – 49%	
E	<40%	

Relationship between post-programming self-efficacy and lower-order engagement constructs

The relationship between *post-programming self-efficacy* and *help-seeking* had a strong positive effect on the average score (*β = .322*) group compared to a negative effect on the good score (*β =-.203*) group. Similarly, the relationship between *post-programming self-efficacy* and *help-seeking* has a strong positive effect on the average score (*β = .322*) group compared to a negative effect on the excellent score (*β =-.201*) group. The relationship between *post-programming self-efficacy* and *surface learning* had a positive effect on the average score (*β =.040*) group compared to a strong negative effect on the excellent score (*β =- .291*) group.

Relationship between lower-order engagement constructs and programming grade

The relationship between *help-seeking* and *programming grade* had a weak positive effect on the average score (*β = .188*) group compared to a negative effect on the excellent score (*β = -.126*) group. The relationship between *effort* and *programming grade* had a strong negative effect on the average score (*β =- .191*) group compared to a strong positive effect on the excellent score (*β = .510*) group. However, when compared to the good score (*β =-.080*) and excellent score (*β = .510*) groups, the effect of *effort* on *programming grade* had a strong positive effect on the excellent programming grade score group compared to a weak negative effect on the good score group.

The relationship between *persistence* and *programming grade* had a weak positive effect on the average score (β = *.138*) group compared to a weak negative effect on the excellent score (β = *-.156*) group. The relationship between *enjoyment* and *programming grade* had a weak negative effect on the average score (β = *-.100*) group compared to a strong positive effect on the excellent score (β = *.268*) group.

Relationship between lower-order engagement constructs and self-assessment

The relationship between *surface learning* and *self-assessment* had a weak positive effect on the average score (β = *.067*) group compared to a strong negative effect on the excellent score (β =- *.200*) group. The relationship between *gratification* and *self-assessment* had a weak positive effect on the good score (β = *.006*) group compared to a strong positive effect on the excellent score (β = *.292*) group. In addition, the relationship between *gratification* and *self-assessment* had a weak positive effect on the average score (β = *.094*) group compared to a strong positive effect on the excellent score (β = *.292*) group.

Table 7.28: Multi-group comparison by Programming Grade

Relationships	Path Coefficient			Confidence Intervals						Comparison	Sig.
				Average		Good		Excellent			
	Average (*n=136*)	Good (*n=135*)	Excellent (*n=162*)	*CI (Low)*	*CI (High)*	*CI (Low)*	*CI (High)*	*CI (Low)*	*CI (High)*		
Post Programming Self-Efficacy -> Help Seeking	0.322***	-0.203*	-0.201	0.196	0.526	-0.410	0.084	-0.390	0.185	Average vs. Good Average vs. Excellent	Sig. Sig.
Post Programming Self-Efficacy -> Surface Learning	0.040	-0.141	-0.291***	-0.252	0.332	-0.387	-0.056	-0.551	-0.217	Average vs. Excellent	Sig.
Help-seeking-> Programming Grade	0.188	-0.087	-0.126	-0.082	0.444	-0.294	0.178	-0.270	0.155	Average vs. Excellent	Sig.
Effort -> Programming Grade	-0.191	-0.080	0.510***	-0.506	0.120	-0.260	0.236	0.320	0.686	Average vs. Excellent Good vs. Excellent	Sig. Sig.
Persistence -> Programming Grade	0.138	0.053	-0.156	-0.122	0.446	-0.163	0.286	-0.343	0.057	Average vs. Excellent	Sig.
Enjoyment -> Programming Grade	0.068	-0.100	0.268**	-0.215	0.278	-0.366	0.225	0.057	0.493	Good vs. Excellent	Sig.
Surface Learning -> Self-Assessment	0.067	-0.177*	-0.200***	-0.097	0.213	-0.300	0.151	-0.307	-0.008	Average vs. Excellent	Sig.
Gratification -> Self-Assessment	0.094	0.006	0.292***	-0.082	0.274	-0.160	0.230	0.103	0.469	Average vs. Excellent Good vs. Excellent	Sig.

Sig. denotes a significance at (p <= 0.05); Nsig. denotes a non-significant relationship
Note: t-values > 1.65 (p<0.1); t-values > 1.96** (p<0.05); t-values > 2.57*** (p<0.01)*

7.11.4 Comparison by Programming Self-Efficacy

The data to compare by programming self-efficacy was grouped based on the difference between the *pre-programming self-efficacy* scores and the *post-programming self-efficacy* scores. The mean scores for *pre-programming self-efficacy* and *post-programming self-efficacy* were computed and the difference in the mean scores was used to group the novice programmers. The differences in the programming self-efficacy scores ranged from *-2.82* to *2.18*. In order to meet the minimum sample size requirement, the scores were divided into two groups. One group was made up of novice programmers whose programming self-efficacy had remained the same or had increased (≥ 0), and the other group was made up of novice programmers whose programming self-efficacy had decreased (< 0). Table 7.29 presents the path coefficients and the significance of the relationships between the groups by using the bias-corrected 95% confidence intervals approach (Shi, 1992).

Relationship between post-programming self-efficacy and lower-order engagement constructs

Interestingly, none of the relationships between the post-programming self-efficacy construct and the lower-order engagement constructs were significantly different between the groups.

Relationship between lower-order engagement constructs and programming grade

The relationship between *persistence* and *programming grade* had a positive effect on the decrease in programming self-efficacy ($\beta = .149$) group compared to a negative effect in the increase in programming self-efficacy ($\beta = -.116$) group.

Relationship between lower-order engagement constructs and self-assessment

Interestingly, none of the relationships between the lower-order engagement constructs and self-assessment were significantly different between the groups.

Table 7.29: Multi-group comparison by Programming Self-Efficacy

Relationships	Path Coefficients		Confidence Intervals				Sig.
	Programming Self-Efficacy						
	Decrease (*n = 164*)	Increase (*n = 269*)	Decrease		Increase		
			CI (Low)	*CI (High)*	*CI (Low)*	*CI (High)*	
Persistence -> Programming Grade	0.149	-0.116	-0.031	0.396	-0.289	-0.023	Sig.

Sig. denotes a significance at (p <= 0.05); Nsig. denotes a non-significant relationship
Note: t-values > 1.65 (p<0.1); t-values > 1.96** (p<0.05); t-values > 2.57*** (p<0.01)*

7.11.5 Comparison by Gender

The data to compare by gender was grouped by males and females. Table 7.30 presents the path coefficients and the significance of the relationships by using the bias-corrected 95% confidence intervals approach (Shi, 1992).

Relationship between post-programming self-efficacy and lower-order engagement constructs

The relationship between *post-programming self-efficacy* and *help-seeking* had a stronger negative effect on the female (*β =-.337*) group compared to the male (*β =-.146*) group. The relationship between *post-programming self-efficacy* and *deep learning* had a stronger positive effect on the male (*β =.731*) group compared to the female (*β =.539*) group.

Relationship between lower-order engagement constructs and programming grade and self-assessment

Interestingly, none of the relationships between the lower-order engagement constructs and programming grade, and between the lower-order engagement constructs and self-assessment were significantly different between the male and female groups.

Table 7.30: Multi-group comparison by Gender

Relationships	Path Coefficients		Confidence Intervals				Sig.
			Male		Female		
	Male (*n = 315*)	Female (*n = 118*)	*CI (Low)*	*CI (High)*	*CI (Low)*	*CI (High)*	
Post-programming self-efficacy -> Help-seeking	-0.146*	-0.337**	-0.306	-0.028	-0.517	-0.243	Sig.
Post-programming self-efficacy -> Deep Learning	0.731***	0.539***	0.682	0.785	0.456	0.696	Sig.

Sig. denotes a significance at (p <= 0.05); Nsig. denotes a non-significant relationship
Note: t-values > 1.65 (p<0.1); t-values > 1.96** (p<0.05); t-values > 2.57*** (p<0.01)*

7.12 Chapter Summary

Although the main purpose of the data analysis was to use the bootstrapping procedure to test the relationship between programming self-efficacy, the lower-order engagement constructs, and programming performance, additional analyses were performed to gain a deeper understanding of the relationships between programming self-efficacy, engagement, and programming performance. Thus, in addition to performing the bootstrapping procedure and testing the hypotheses, additional analyses such as the IPMA, the significance of the confounding variables, the significance of the higher-order engagement constructs, multiple mediation and MGA were also performed. The results of the analysis showed a detailed examination of the relationships between programming self-efficacy, engagement and the programming performance of novice programmers in introductory programming courses.

Chapter 8: Discussion of Findings

The discussion of the findings from the survey questionnaires (Phase 1 and Phase 3) are organised by Research Question (RQ1), followed by Research Question 2 (RQ2). The findings of the confounding variables in this research are then discussed and the final research model for this study is presented.

8.1 Programming self-efficacy beliefs before and after instruction

The discussion of the findings in this section partially answers RQ1: *What is the effect of self-efficacy on the novice programmer's engagement in an introductory programming course?*

This study examined the change in the programming self-efficacy beliefs of the novice programmers at the beginning and at the end of the introductory programming course by adopting Ramalingam and Wiedenbeck's (1998) programming self-efficacy scale, and the academic self-efficacy scales by Bresó et al. (2011), Sherer et al. (1982), and Walker et al. (2006).

8.1.1 Pre-programming self-efficacy

Pre-programming self-efficacy was previously defined as "the belief of the novice programmer in performing programming related tasks at the beginning of the introductory programming course" (Chapter 3, Section 3.2.2). The data for the pre-programming self-efficacy beliefs was collected in the third week of the introductory programming course. The higher-order pre-programming self-efficacy construct contained five reflectively measured constructs which are *general self-efficacy* (6 items), *independence and persistence* (7 items), *complex programming tasks* (5 items), *self-regulation* (2 items), and *simple programming tasks* (5 items). An Exploratory Factor Analysis (EFA) was performed on the construct during the pilot study phase only since the construct had been validated in prior research. During the Confirmatory Factor Analysis (CFA) of the measurement model, 11 items were deleted in order to satisfy the convergent validity of the pre-programming self-efficacy construct. Table 8.1 presents the final set of items that was used to measure pre-programming self-efficacy. The construct satisfied all validity and reliability criteria (α = .924; CR = .934; AVE = .505).

Table 8.1: Items to measure pre-programming self-efficacy

Construct:		**Pre-programming self-efficacy**
Factor	**Code**	**Item**
Note: All indicators start with *"I am confident that……."*		
General self-efficacy	**BPSE1**	I can complete a new task that is assigned to me if I keep trying.
	BPSE6	I have the capability to learn the contents of this programming course.
	BPSE12	I can find ways of overcoming the problem if I get stuck at a point while working on a programming assessment.
	BPSE13	I can correct (debug) all the errors in a program that I have written, and make it work.
Complex programming tasks	**BPSE14**	I can apply the right programming concepts to solve a problem given to me.
	BPSE15	I can understand and apply the fundamental object-oriented programming concepts used in my programming course.
	BPSE16	I can make use of a pre-written library in the programming integrated development environment (IDE).
	BPSE17	I can write a program to solve any given problem as long as the specifications are clear.
	BPSE18	I can mentally trace through the execution of a complex program given to me.
Simple programming tasks	**BPSE21**	I can write programming code that runs without errors (no syntax errors)
	BPSE22	I can write programming code that is logical
	BPSE23	I can write a small program for a small problem that is familiar to me
	BPSE24	I can understand the language structure and the usage of the reserved words /keywords in the programming language.
	BPSE25	I can write a reasonably sized program that can solve a problem that is only vaguely familiar to me.

8.1.2 Post-programming self-efficacy

Post-programming self-efficacy was previously defined as "the belief of the novice programmer in performing programming related tasks at the end of the introductory programming course" (Chapter 3, Section 3.2.2). The data for the post-programming self-efficacy beliefs was collected at the end of the introductory programming course. The scale used to collect data for the pre-programming self-efficacy beliefs was repeated at the end of the introductory programming course. The participants were asked to reflect on their self-efficacy beliefs in performing programming-related tasks during their introductory programming course. Like the pre-programming self-efficacy construct, the items in this construct were examined using an EFA during the pilot study phase. During the CFA of the measurement model, 6 items were deleted in order to satisfy the convergent validity of the post-programming self-efficacy construct.

Table 8.2 presents the final set of items that were used to measure post-programming self-efficacy. The construct satisfied all validity and reliability criteria (α = .951; CR = .956; AVE = .535).

Table 8.2: Items to measure post-programming self-efficacy

Construct:		**Pre-programming self-efficacy**
Factor	**Code**	**Item**
Note: All indicators start with *"I am confident that……."*		
General self-efficacy	**APSE1**	I can complete a new task that is assigned to me if I keep trying.
	APSE2	I can stick to completing a task even though it may be unpleasant.
	APSE4	When unexpected problems occur I can handle them well.
	APSE5	I can achieve the goals that I set for myself in this programming course.
	APSE6	I have the capability to learn the contents of this programming course.
Independence and persistence	**APSE7**	I can complete my programming assessment if I had a lot of time.
	APSE12	I can find ways of overcoming the problem if I get stuck at a point while working on a programming assessment.
	APSE13	I can correct (debug) all the errors in a program that I have written, and make it work.
Complex programming tasks	**APSE14**	I can apply the right programming concepts to solve a problem given to me.
	APSE15	I can understand and apply the fundamental object-oriented programming concepts used in my programming course.
	APSE16	I can make use of a pre-written library in the programming integrated development environment (IDE).
	APSE17	I can write a program to solve any given problem as long as the specifications are clear.
	APSE18	I can mentally trace through the execution of a complex program given to me.
	APSE20	I can find ways of motivating myself to program, even if the problem area was of no interest to me.
Simple programming tasks	**APSE21**	I can write programming code that runs without errors (no syntax errors).
	APSE22	I can write programming code that is logical.
	APSE23	I can write a small program for a small problem that is familiar to me.
	APSE24	I can understand the language structure and the usage of the reserved words /keywords in the programming language.
	APSE25	I can write a reasonably sized program that can solve a problem that is only vaguely familiar to me.

There was a strong positive and statistically significant relationship between pre- and post-programming self-efficacy (*β=.568*). Thus, **Hypothesis (H1):** Pre-programming self-efficacy belief will have a positive effect on post-programming self-efficacy belief is **SUPPORTED**. The strong positive relationship between pre- and post-programming self-efficacy supports the findings from prior research which found that the self-efficacy beliefs of novice programmers increased significantly as they progressed in their introductory programming course (Ramalingam et al., 2004; Wiedenbeck, 2005).

Interestingly, the group-specific results revealed that the relationship between pre- and post-programming self-efficacy was significantly stronger in the novice programmers with an average (*β=.711*) intelligence score compared to the novice programmers with a good (*β=.501*) or excellent (*β=.482*) intelligence score. A plausible explanation for this result is offered in Chapter 9, Section 9.2.

Although not a direct predecessor of the dependent variables *programming grade* and *self-assessment*, *post-programming self-efficacy* had ranked highest in its performance and importance in the IPMA (Chapter 7, Section 7.7) compared to the engagement constructs that were direct predecessors of the dependent variables. This finding suggests that the programming self-efficacy beliefs of the novice programmer is an important indicator for programming success, and a recommendation is made in Chapter 9, Section 9.7.1 for course instructors to build the self-efficacy beliefs of the novice programmer.

The analysis for multiple mediation showed a strong positive effect between post-programming self-efficacy and the dependent variables *programming grade* (*β=.416*) and *self-assessment* (*β=.695*) (Chapter 7, Section 7.10 and Appendix G). This finding is in line with prior literature that showed a direct relationship between self-efficacy and performance (Joo, Lim, & Kim, 2013; Phan, 2011; Pintrich & Schunk, 1996; Zimmerman, 2000). However, other studies have argued that self-efficacy is a predictor of behaviour instead of performance (Bandura, 1977; Bandura, 2012; Joo et al., 2013; Pintrich & Schunk, 1996; Yip, 2012; Zimmerman, 2000), which is the focus of this research. Therefore, the direct relationship between self-efficacy and performance will not be discussed further since the aim of this study is to examine the engagement construct by examining the effect of self-efficacy on engagement, and the effect of engagement on the performance of the novice programmer.

8.2 The effect of programming self-efficacy on engagement

The discussion of the findings in this section answers RQ1: *What is the effect of self-efficacy on the novice programmer's engagement in an introductory programming course?* The discussion is organised by the three higher-order engagement constructs: behavioural engagement, cognitive engagement, and emotional engagement. The effect of programming self-efficacy on each indicator of the higher-order engagement constructs is then discussed individually.

In Phase 3 of this research, this study examined the engagement behaviour of the novice programmers in their introductory programming course by using a survey questionnaire that was developed using a three stage instrument development process that was proposed by Moore & Benbasat (1991) and was subjected to rigorous reliability and validity tests. The Ramalingam and Wiedenbeck's (1998) programming self-efficacy scale was also repeated in Phase 3 of this study.

8.2.1 Behavioural Engagement

In Chapter 3, Section 3.2.3.2, the higher-order behavioural engagement construct was defined as the student's involvement in observable academic, social and extracurricular activities (Fredricks et al., 2004; Yazzie-Mintz & McCormick, 2012). The behavioural engagement construct was formatively measured by the lower-order engagement constructs -*effort*, *persistence*, and *help-seeking*. In addition, the *participation* construct was also proposed as a measure of behavioural engagement in the early stages of this study, but was dropped from the research model during the scale development phase due to lack of reliable measures.

The higher-order behavioural engagement construct satisfied all validity criteria (Chapter 7, Section 7.9.1). Post-programming self-efficacy moderately predicted 51.1% of the variance in behavioural engagement ($R^2 = .511$). **Hypothesis HPSEB:** Self-efficacy beliefs in learning programming will have a positive effect on behavioural engagement is **SUPPORTED** since there was a strong positive and statistically significant relationship between post-programming self-efficacy and behavioural engagement ($\beta=.715$).

The analysis for multiple mediation showed that the higher-order behavioural engagement construct mediated the relationship between post-programming self-efficacy and the participant's self-assessment of their programming performance (Chapter 7, Section 7.10 & Appendix G). The mediating relationship will not be examined further since the focus of this study is to examine the relationship between programming self-efficacy and engagement, and the relationship between engagement and the programming performance of the novice programmer.

8.2.1.1 Help-seeking

In Chapter 3, Section 3.2.3.2, help-seeking was defined as novice programmers with "an achievement behavior involving the search for, and employment of a strategy to obtain success" (Ames & Lau, 1982, p. 414). Help-seeking was initially measured by 5 items but reduced to 3 items in the final set after being assessed for validity and reliability. The two items that were deleted were related to help-seeking threat and help-seeking avoidance strategies. The final three items (BEHS1, BEHS3, and BEHS5) measured instrumental help-seeking strategies. Table 8.3 presents the final set of items that was used to measure help-seeking. The help-seeking construct satisfied all validity and reliability criteria ($\alpha = .659$; CR = .762; AVE = .535). Post-programming self-efficacy predicted 3.7% of the variance in help-seeking ($R^2 = .037$) which is less than the acceptable threshold suggested by Chin (1998).

The items to measure help-seeking were developed from the findings of the focus groups. The participants explained that when they were stuck in their programming assessment, they often asked for help from their course instructors, peers, family members, and referred to secondary resources such as textbooks or the Internet (Chapter 5, Section 5.3.2.2). In addition, the items were also developed based on previous literature with an emphasis on the instrumental help-seeking behaviour.

Table 8.3: Items to measure help-seeking

Construct:	**Help-seeking**
Code	**Item**
BEHS1	Asking for help would be one of the first things I would do if I had errors in my programming assessment.
BEHS3	I get help whenever I can to get through my programming assessments.
BEHS5	I expect to be given a hint to the solution when I ask for help to debug my programming errors.

The relationship between post-programming self-efficacy and help-seeking was initially hypothesised as a positive relationship. However, a moderate negative but statistically significant relationship was observed between post-programming self-efficacy and help-seeking (β=-.*192*). Since the result of the relationship was negative, **Hypothesis (H3):** Self-efficacy beliefs in learning programming will have a positive effect on help-seeking is **NOT SUPPORTED**.

Clearly, the negative relationship between post-programming self-efficacy and help-seeking contradicts the findings from prior research that found a positive relationship between self-efficacy and instrumental help-seeking behaviour (Ryan & Pintrich, 1998; Ryan & Shin, 2011). The group-specific results may offer a plausible explanation. Firstly, a moderately strong positive relationship was observed between post-programming self-efficacy and the need to seek help in the Malaysia group (β=.*317*) compared to a moderately strong negative relationship in the New Zealand group (β=-.*276*). While the finding in the Malaysia group was in line with the findings from previous research, the same conclusion could not be reached with the New Zealand group.

In the second group-specific result, there was a moderately strong positive relationship between post-programming self-efficacy and the need to seek help in the novice programmers with an average programming grade (β=.*322*) while there was a moderately strong but negative and statistically weak to not significant relationship between post-programming self-efficacy and the need to seek help in the novice programmers with a good (β=-.*203*), and excellent (β=-.*201*) programming grade. This second finding suggests that the average performers were less reluctant to ask for help to resolve their programming problems as their self-efficacy beliefs increased. Secondly, although not a statistically significant

relationship, novice programmers who performed well in the course and had higher self-efficacy beliefs in succeeding in the introductory programming course asked for less help compared to their peers who did not perform as well.

The third group-specific observation showed that the tendency to ask for help decreased significantly with an increase in post-programming self-efficacy beliefs in the female novice programmers (β=-.*337*) compared to the male novice programmers (β=-.*146*).

8.2.1.2 Effort

In Chapter 3, Section 3.2.3.2, effort was defined as the overall amount of effort expended in the process of studying (Zimmerman & Risemberg, 1997). Effort was initially measured by 6 items and reduced to 4 items after being assessed for validity and reliability. Table 8.4 presents the final set of items that was used to measure effort. The effort construct satisfied all validity and reliability criteria (α = .701; CR = .808; AVE = .517). Post-programming self-efficacy predicted 31.3% of the variance in effort (R^2 = *.313*).

The items to measure effort were developed from the findings of the focus groups. The participants explained that frequent practise was essential when learning programming. The participants also planned their time well when attempting their programming assessments, and consistently worked on their programming assessments throughout the course (Chapter 5, Section 5.3.2.3).

Table 8.4: Items to measure effort

Construct:	**Effort**
Code	**Item**
BEEF2	I try to stay on top of my programming assessments by starting and completing them early.
BEEF5	I often practise writing programs in order to understand a programming concept.
BEEF6	I organise my time effectively in order to maximise the effort spent on my programming assessment.
BEPA1	I worked all the way through the course to complete all of my programming assessments.

There was a strong positive and statistically significant relationship between post-programming self-efficacy and effort (β=*.560*). Thus, **Hypothesis (H4):** Self-efficacy beliefs in learning programming will have a positive effect on effort is **SUPPORTED**. The strong positive relationship between post-programming self-efficacy and effort supports the findings from previous research on learning theory that found that self-efficacy influences effort (Bandura, 1977; Pintrich & Schunk, 1996; Zimmerman, 2000), and more recently, Bandura (2012) explained that individuals "of low self-efficacy are easily convinced of the futility of effort when they come up against institutional impediments, whereas those of high self-

efficacy figure out ways to surmount them" (p. 14). However, in the published literature on computer programming, there appears to be no evidence of the effect of self-efficacy on the novice programmer's effort when learn programming.

8.2.1.3 Persistence

In Chapter 3, Section 3.2.3.2, persistence was defined as a student's continued undertaking of an academic task despite facing obstacles or setbacks (Bye et al., 2007). Persistence was initially measured by 5 items and remained at 5 items in the final set after being assessed for validity and reliability. However, item BEPE2* (negatively worded item) was deleted while item BEEF3 was moved to the persistence construct during the first card sorting round. Table 8.5 presents the final set of items that was used to measure persistence. The persistence construct satisfied all validity and reliability criteria (α = .767; CR = .842; AVE = .519). Post-programming self-efficacy moderately predicted 40.0% of the variance in persistence ($R^2 = .400$).

The items to measure persistence were developed from the findings of the focus groups. The participants explained that learning programming required determination, persistence in debugging errors in the programming code, and perseverance by practising on writing code for programming problems (Chapter 5, Section 5.3.2.4).

Table 8.5: Items to measure persistence

Construct:	**Persistence**
Code	**Item**
BEPE1	I can spend hours debugging an error in my programming assessment without feeling like giving up.
BEPE3	I review my lecture notes and refer to other resources again and again in order to try and understand a difficult programming concept or problem.
BEPE4	When I don't understand a programming problem, I keep practising until I understand the problem.
BEPE5	When my program doesn't work, I am determined to fix the errors no matter how long it takes.
BEEF3	When I don't understand a programming concept, I take time to review my lecture notes and online resources that would help me understand the concept.

There was a strong positive and statistically significant relationship between post-programming self-efficacy and persistence (β=.633). Thus, **Hypothesis (H5):** Self-efficacy beliefs in learning programming will have a positive effect on persistence is **SUPPORTED**. The strong positive relationship between post-programming self-efficacy and persistence supports the findings from previous research on learning theory that self-efficacy influences persistence (Bandura, 1977; Brown et al., 2008; Pintrich & Schunk, 1996; Zimmerman, 2000). However, in the published literature on computer programming, there appears to be no

evidence of the effect of self-efficacy on the novice programmer's persistence when learning programming.

One finding from the group-specific analysis revealed that novice programmers with an average intelligence score (β=*.702*) showed a significantly stronger positive relationship between post-programming self-efficacy and persistence compared to the novice programmers with a good intelligence score (β=*.539*).

8.2.2 Cognitive Engagement

In Chapter 3, Section 3.2.3.3, the higher-order cognitive engagement construct was defined as the student's investment in learning, motivation to learn, and use of strategies for learning (Sheard et al., 2010; Yazzie-Mintz & McCormick, 2012). The cognitive engagement construct was formatively measured by the lower-order engagement constructs -*deep learning*, *surface learning*, and *trial and error*.

The higher-order cognitive engagement construct satisfied all validity criteria (Chapter 7, Section 7.9.1). **Hypothesis HPSEC:** Self-efficacy beliefs in learning programming will have a positive effect on cognitive engagement is **SUPPORTED** since there was a strong positive and statistically significant relationship between post-programming self-efficacy and cognitive engagement (β=*.725*). Post-programming self-efficacy construct moderately predicted 52.5% of the variance in cognitive engagement (R^2 = *.525*).

8.2.2.1 Deep Learning

In Chapter 2, Section 2.1.4.2, deep learning was conceptualised as having intrinsic value to the student, adds meaning to the learning task by connecting knowledge of what is known to new knowledge resulting in improved retention (Biggs, 1987; Marton & Säljö, 1976; Ramsden, 2003). Deep learning was initially measured by 4 items but increased to 5 items in the final set because item CETE1 had loaded strongly on the deep learning construct during the EFA of the actual study. Table 8.6 presents the final set of items that was used to measure deep learning. The deep learning construct satisfied all validity and reliability criteria (α = .757; CR = .837; AVE = .508). Post-programming self-efficacy moderately predicted 45.7% of the variance in deep learning (R^2 = *.457*).

The items to measure deep learning were developed from the findings of the focus groups and from the published literature on learning theory. The focus group participants explained that they were engaged in learning tasks such as analysing programming code, and understanding the logic of the programming code (Chapter 5, Section 5.3.3.1).

There was a strong positive and statistically significant relationship between post-programming self-efficacy and deep learning (β=*.676*). Thus, **Hypothesis (H6):** Self-efficacy beliefs in learning programming will have a positive effect on deep learning approaches is **SUPPORTED**. The positive relationship between post-programming self-efficacy and deep learning supports the findings from the published literature on learning

theory that found a positive relationship between self-efficacy and the use of a deep learning approach to learn (Fenollar et al., 2007; Phan, 2011; Pintrich & De Groot, 1990; Schunk & Mullen, 2012).

Table 8.6: Items to measure deep learning

Construct:	**Deep Learning**
Code	**Item**
CEDL1	When I am given a programming problem to solve, I try to understand the program logic before I start coding the solution.
CEDL2	I carefully read through the assessment criteria for my programming assessment so that I know what I have to do in order to succeed in my programming course.
CEDL3	I find that programming problems often stir my curiosity and make me think deeply about how to code the solutions to the problems.
CEDL4	When I encounter an error in my programming assessment, I try to work out why my piece of code didn't work by stepping through the logic of the code that I had written.
CETE1	If I get a logic error in my programming assessment, I debug my code by stepping through the code line by line.

The group-specific results revealed a significant difference between the genders in the relationship between post-programming self-efficacy and deep learning. The male novice programmers had a stronger positive relationship between post-programming self-efficacy and the use of a deep learning approach ($\beta=.731$) compared to the female novice programmers ($\beta=.539$).

8.2.2.2 Surface Learning

In Chapter 2, Section 2.1.4.2, surface learning was conceptualised as having extrinsic value to the student. A surface learner focuses on rote learning, memorizes concepts, completes a task without giving any meaning to the task, and does not attempt to reflect on the concepts or facts, and their relation to prior knowledge (Biggs, 1987; Marton & Säljö, 1976; Ramsden, 2003). Surface learning was initially measured by 8 items but reduced to 4 items in the final set after being assessed for validity and reliability. Table 8.7 presents the final set of items that was used to measure surface learning. The surface learning construct satisfied all validity and reliability criteria (α = .695; CR = .805; AVE = .515). Post-programming self-efficacy predicted only 7.4% of the variance in surface learning ($R^2 = .074$) which was less than the acceptable threshold suggested by Chin (1998).

The items to measure surface learning were developed from the findings of the focus groups and from the published literature on learning theory. The focus group participants explained that their learning approach involved memorizing code, preparing for lessons only when required, and referred to their lecture slides for answers to their programming problems (Chapter 5, Section 5.3.3.2).

Table 8.7: Items to measure surface learning

Construct:	**Surface Learning**
Code	**Item**
CESL2	When working on my programming assessment, I re-use the code from my course notes or from other resources although I am not sure how the code works.
CESL3	Once I get help with solving a programming problem, I move on and do not wonder why the error occurred.
CESL5	I find that I often rely on my friend's suggestions to make my programming code work.
CESL6	When I encounter an error in my programming assessment, I fix the error without actually understanding where my program went wrong.

There was a moderately strong negative and statistically significant relationship between post-programming self-efficacy and surface learning (β=-.*273*). Thus, **Hypothesis (H7):** Self-efficacy beliefs in learning programming will have a negative effect on surface learning approaches is **SUPPORTED**. The negative relationship between post-programming self-efficacy and surface learning supports the findings from previous research on learning theory which found that students with high self-efficacy beliefs tended to avoid using a surface learning approach to learn (Fenollar et al., 2007; Phan, 2011; Pintrich & De Groot, 1990; Schunk & Mullen, 2012).

In addition, the group-specific results revealed a moderately strong negative relationship between post-programming self-efficacy and the use of a surface learning approach in the New Zealand group (β=-.*341*) compared to a weak negative but not statistically significant relationship in the Malaysia group (β=-.*050*).

Another group-specific result revealed a moderately strong negative and statistically significant relationship between post-programming self-efficacy and the use of a surface learning approach in the novice programmers who obtained an excellent programming grade (β=-.*291*) compared to a weak positive but not statistically significant relationship in the novice programmers who had obtained an average programming grade (β=.*040*).

8.2.2.3 Trial and Error (New Construct)

In Chapter 3, Section 3.2.3.3 trial and error was proposed as a new indicator of cognitive engagement in introductory programming courses based on the evidence from prior literature on computer programming which suggested that programming students used a trial and error approach to debug the errors in their programming code (Blikstein, 2011; Dorn & Guzdial, 2010; Ebrahimi, 2012; Edwards, 2004). Trial and error was defined as "trying out new strategies, rejecting choices that are erroneous in the sense that they do not lead to higher payoffs" (Young, 2009, p. 626). Trial and error was initially measured by 5 items but reduced to 4 items in the final set after being assessed for validity and reliability. Table 8.8 presents the final set of items that was used to measure trial and error. The trial and error construct satisfied all validity and reliability criteria (α = .723; CR = .825; AVE = .548). Post-programming self-efficacy weakly predicted 26.6% of the variance in trial and error (R^2 = *.266*).

The items to measure trial and error were developed from the definition of the construct, and from the findings of the focus groups. The focus group participants explained that debugging programming errors required a trial and error strategy (Chapter 5, Section 5.3.3.3).

Table 8.8: Items to measure trial and error

Construct:	**Trial and Error**
Code	**Item**
CETE2	I try different solutions when I work on practise problems.
CETE3	I often try different ways to solve my programming problem.
CETE5	When I work on a programming problem, I move the codes around to make it work.
CESL4	When I encounter an error in my programming assessment, I try to debug the error by inserting any code which looks right to me.

There was a strong positive and statistically significant relationship between post-programming self-efficacy and trial and error (β=*.516*). Thus, **Hypothesis (H8):** Self-efficacy beliefs in learning programming will have a positive effect on trial and error is **SUPPORTED**. The strong positive relationship between post-programming self-efficacy and trial and error provided strong evidence that trial and error is a new indicator of cognitive engagement and that it appears to be an important engagement factor in the programming discipline although no published research could be found to support this relationship.

The group-specific results revealed a stronger positive relationship between post-programming self-efficacy and the use of a trial and error approach in the novice programmers with an average intelligence score (β=*.601*) compared to the novice programmers with a good intelligence score (β=*.373*). Similarly, novice programmers with an excellent intelligence score (β=*.639*) showed a stronger positive relationship between post-programming self-efficacy and the use of a trial and error approach compared to the

novice programmers with a good intelligence score (*β=.373*). This group-specific finding suggests that novice programmers with an average or excellent intelligence score appear to have the tendency to use a trial and error strategy to solve their programming problems, particularly to debug their programming errors.

8.2.3 Emotional Engagement

In Chapter 3, Section 3.2.3.4, the higher-order emotional engagement construct was defined as the student's feeling, attitude, and perception towards learning, and the learning environment (Sheard et al., 2010; Yazzie-Mintz & McCormick, 2012) and is "presumed to create ties to an institution and influence willingness to do the work" (Fredricks et al., 2004, p. 60). The emotional engagement construct was formatively measured by the lower-order engagement constructs - *enjoyment*, *interest*, and *gratification*.

The higher-order emotional engagement construct satisfied all validity criteria (Chapter 7, Section 7.9.1). There was a strong positive and statistically significant relationship between post-programming self-efficacy and emotional engagement (*β=.654*). Thus, **Hypothesis HPSEE:** Self-efficacy beliefs in learning programming will have a positive effect on emotional engagement is **SUPPORTED.** Post-programming self-efficacy moderately predicted 42.8% of the variance in emotional engagement ($R^2 = .428$).

One observation from the multiple mediation test showed that the higher-order emotional engagement construct mediated the relationship between post-programming self-efficacy and the participant's self-assessment of their programming performance (Chapter 7, Section 7.10 & Appendix G). This mediating relationship will not be examined further since the focus of this study is to examine the relationship between programming self-efficacy and engagement, and the relationship between engagement and the programming performance of the novice programmer.

8.2.3.1 Enjoyment

In Chapter 3, Section 3.2.3.4, enjoyment was defined as novice programmers who focus their "attention to its object or an experience…and to have one's desires satisfied while doing" a task (White, 1964, p. 325-326). On the other hand, Davis (1982) explained that enjoyment "causes the subject to experience pleasure by causing occurrent beliefs which satisfy desires concerning the experience itself" (p. 240). Enjoyment was initially measured by 4 items but reduced to 3 items in the final set after being assessed for validity and reliability. Table 8.9 presents the final set of items that was used to measure enjoyment. The enjoyment construct satisfied all validity and reliability criteria (α = .701; CR = .832; AVE = .625). Post-programming self-efficacy moderately predicted 39.0% of the variance in enjoyment ($R^2 = .390$).

The items to measure enjoyment were developed from the published literature on computer programming and from the findings of the focus groups. The focus group participants

explained that writing solutions to programming problems was appealing to them and that they enjoyed programming (Chapter 5, Section 5.3.4.1).

Table 8.9: Items to measure enjoyment

Construct:	**Enjoyment**
Code	**Item**
EEEN1	My programming course is stimulating compared to the other courses that I am enrolled in.
EEEN2	I like writing programs.
EEEN3	The challenge of coding solutions to programming problems appeals to me.

There was a strong positive and statistically significant relationship between post-programming self-efficacy and enjoyment (β=*.625*). Thus, **Hypothesis (H10):** Self-efficacy beliefs in learning programming will have a positive effect on enjoyment is **SUPPORTED**. The positive relationship between post-programming self-efficacy and enjoyment supports the finding from the published literature on learning theory that found positive correlations between self-efficacy and enjoyment (Mills et al., 2007), and that high self-efficacy results in pleasant emotions such as enjoyment (Pekrun et al., 2004; Putwain et al., 2013).

The group-specific results revealed a stronger positive relationship between post-programming self-efficacy and the feeling of enjoyment in the novice programmers with an average intelligence score (β=*.686*) compared to the novice programmers with a good intelligence score (β=*.586*).

8.2.3.2 Gratification (New Construct)

In Chapter 5, Section 5.3.4.3, gratification was proposed as a new indicator of emotional engagement since the findings from the focus groups showed that the novice programmers experienced a feeling of being rewarded and had looked forward to seeing the immediate output upon successfully coding a solution to their programming problem. Gratification was then defined as novice programmers who experience "the feeling of receiving immediate rewards typically in the form of pleasure or satisfaction as a result of hard work" (Chapter 5, Section 5.3.4.3). The interpretation of gratification in this study focuses on *immediate gratification* as opposed to *delay of gratification* which is a common outcome of learning in the literature on learning theory.

Gratification was initially measured by 5 items but reduced to 4 items in the final set after being assessed for validity and reliability. Table 8.10 presents the final set of items that was used to measure gratification. The gratification construct satisfied all validity and reliability criteria (α = .768; CR = .851; AVE = .592). Post-programming self-efficacy weakly predicted 22.1% of the variance in gratification (R^2 = *.221*).

The items to measure gratification were developed from the findings of the focus groups, whereby participants explained that they felt rewarded when they successfully debugged their errors and were able to see the output of their program (Chapter 5, Section 5.3.4.3).

Table 8.10: Items to measure gratification

Construct:	**Gratification**
Code	**Item**
EEGR1	I feel a deep sense of satisfaction when I finally get my program to work.
EEGR2	I feel relieved when I complete my programming assessment.
EEGR4	I feel a strong sense of achievement when I complete my programming assessments.
EEGR5	Once I complete my programming assessment, I feel pleased that I have successfully completed a challenging piece of assessment.

There was a moderately strong positive and statistically significant relationship between post-programming self-efficacy and gratification (β=*.470*). Thus, **Hypothesis (H20):** Self-efficacy beliefs in learning programming will have a positive effect on gratification is **SUPPORTED**.

The positive relationship between post-programming self-efficacy and gratification provided strong evidence that gratification is a new indicator of emotional engagement and that it appears to be an important engagement factor in the programming discipline although no published research could be found to support this relationship.

8.2.3.3 Interest

In Chapter 3, Section 3.2.3.4, interest was defined as "an emotion that arouses attention to, curiosity about, and concern with…" a discipline of study (Akbulut & Looney, 2007, p. 68). Interest was initially measured by 5 items but reduced to 3 items in the final set after being assessed for validity and reliability. Table 8.11 presents the final set of items that was used to measure interest. The interest construct satisfied all validity and reliability criteria (α = .766; CR = .862; AVE = .677). Post-programming self-efficacy weakly predicted 31.8% of the variance in interest (R^2 = *.318*).

The items to measure interest were developed from the findings of the focus groups. The focus group participants showed their interest in learning programming by using words such as "passionate", "cool", "stirs my curiosity", "vocation", and had intended to take up programming as a career (Chapter 5, Section 5.3.4.2).

Table 8.11: Items to measure interest

Construct:	**Interest**
Code	**Item**
EEIN1	My programming assessments take priority over the assessments from other courses.
EEIN3	I am passionate about programming.
EEIN4	My programming course suits me better than the other courses that I am currently enrolled in.

There was a strong positive and statistically significant relationship between post-programming self-efficacy and interest (β=*.564*). Thus, **Hypothesis (H9):** Self-efficacy beliefs in learning programming will have a positive effect on interest is **SUPPORTED**. The positive relationship between post-programming self-efficacy and interest supports the findings from the published research on learning theory and computer programming which found that self-efficacy influences interest (Bandura, 1997; Linnenbrink & Pintrich, 2003; Rottinghaus et al., 2003; Silvia, 2003; Wiedenbeck et al., 2007).

8.3 The effect of engagement on programming performance

The discussion of the findings in this section answers RQ2*: What is the effect of the novice programmer's engagement on their performance in an introductory programming course?*

Programming performance which is the dependent variable in this study is measured by two constructs: *programming grade* and *self-assessment*. The effect of the lower-order engagement constructs on programming grade and self-assessment were largely similar, with the exception of one difference observed in the relationship between gratification and programming performance.

8.3.1 Programming Grade

The programming grade construct was measured using a single item scale - the final course grade of the novice programmer. The lower-order engagement constructs weakly predicted 25.0% of the variance in programming grade (R^2 = *.250*).

8.3.1.1 Engagement constructs that predicted programming grade

The higher-order behavioural engagement and cognitive engagement constructs predicted programming grade. There was a weak positive relationship between behavioural engagement and programming grade (β=*.181*), and there was a moderately strong positive relationship between cognitive engagement and programming grade (β=*.210*). Specifically, the lower-order engagement constructs - *effort*, *deep learning*, *surface learning*, and *enjoyment* predicted programming grade.

Effort

There was a moderately strong positive and statistically significant relationship between effort and programming grade (*β=.230*). **Hypothesis (13a):** Effort in learning programming will have a positive effect on course grade is **SUPPORTED**. The positive relationship between effort and programming grade supports the findings from the published literature on learning theory and computer programming which found that effort influences performance (Diseth et al., 2010; Dupeyrat & Mariné, 2005; Liu et al., 2009; McClure et al., 2011; McKinney & Denton, 2004; Ventura, 2005; Weiner, 1985), and in a recent study, Komarraju and Nadler (2014) confirmed the relationship between self-efficacy, effort and the achievement of grades.

Interestingly, when compared to the other indicators of engagement, effort was ranked the second highest in its performance and importance in the IPMA (Chapter 7, Section 7.7). This finding suggests that effort is an important predictor of programming success, and a recommendation is made in Chapter 9, Section 9.7.1 for course instructors to tactfully enable the novice programmers to expend effort when learning programming.

Although the relationship between effort and programming grade is strong, the group-specific results revealed two significant differences. The first group-specific result showed a strong positive relationship between effort and programming grade in the New Zealand group (*β=.413*) compared to a weak negative and not statistically significant relationship in the Malaysia group (*β=-.017*).

Another group-specific result revealed a strong positive relationship between effort and programming grade in the novice programmers with an excellent programming grade (*β=.510*) compared to a negative and not statistically significant relationship in the novice programmers with an average (*β=-.191*) or good (*β=-.080*) programming grade.

Deep Learning

There was a weak positive but statistically significant relationship between deep learning and programming grade (*β=.129*). This relationship was significant at a 5% probability of error. **Hypothesis (15a):** A deep learning approach in learning programming will have a positive effect on course grade is **SUPPORTED**. Surprisingly, the weak positive relationship between deep learning and programming grade was not consistent with the findings from previous research on learning programming that found a strong correlation between deep learning and performance (de Raadt et al., 2005; Diseth et al., 2010; Miller et al., 1996; Simon et al., 2006; Yip, 2012).

On the other hand, Hughes and Peiris (2006) found a weak positive relationship between deep learning and programming grade. Hughes and Peiris (2006) argued that their finding was not surprising due to two reasons. First, the novice programmers who used a deep

learning approach may not have clearly understood the requirements of the programming assessment, and second, the programming assessment may not be assessing the understanding of the student but instead rewarding students who take a strategic approach to learning (Hughes & Peiris, 2006, p. 277).

In addition, there were two findings in the published literature on learning theory that was similar to the finding in this study. Campbell and Cabrera (2014) found that deep learning was not related to grades while Diseth et al. (2010) found that the positive effect of deep learning on achievement had reduced significantly when surface and strategic learning approaches were controlled. As a result, they suggested that a surface learning approach should be discouraged if the intention is to improve performance instead of encouraging a deep learning approach.

Surface Learning

There was a moderately strong negative and statistically significant relationship between surface learning and programming grade (β=-.235). **Hypothesis (16a):** A surface learning approach in learning programming will have a negative effect on course grade is **SUPPORTED**. The relationship between surface learning and programming grade supported the findings from the published literature on learning theory and computer programming which found that surface learning affects performance (de Raadt et al., 2005; Diseth et al., 2010; Hughes & Peiris, 2006; Miller et al., 1996; Simon et al., 2006; Yip, 2012).

Not surprisingly, when compared with the other engagement constructs, surface learning was ranked the lowest in its performance and importance in the IPMA (Chapter 7, Section 7.7). This suggests that using a surface learning approach leads to negative outcomes and is a cognitive engagement strategy that should be avoided.

Enjoyment

There was a moderately strong positive and statistically significant relationship between enjoyment and programming grade (β=.266). **Hypothesis (19a):** Enjoyment in learning programming will have a positive effect on course grade is **SUPPORTED**. The positive relationship between enjoyment and programming grade supports the findings from the published literature on learning theory that found positive correlations between enjoyment and performance (Frenzel et al., 2007; Pekrun et al., 2002). In addition, the recently published literature on computer programming found a relationship between enjoyment and learning programming when learning interventions such as pair programming (Liebenberg et al., 2012; Maguire, Maguire, Hyland, & Marshall, 2014), or new programming tools were introduced (Bishop-Clark et al., 2007; Major, Kyriacou, & Brereton, 2014). However, these studies did not examine the relationship between enjoyment and programming performance.

When compared with the other engagement constructs, enjoyment was ranked the highest in its performance and importance in the IPMA (Chapter 7, Section 7.7). This finding suggests that enjoyment is an important predictor of programming success, and a recommendation is made in Chapter 9, Section 9.7.1 for course instructors to develop enjoyable programming tasks for novice programmers who are learning programming.

Based on the evidence from the focus groups, the novice programmers who appear to enjoy programming have an innate liking to write programs, and to apply logical thinking skills to solve their programming problems. Based on evidence from the existing literature on computer programming, introducing learning interventions such as pair programming (Liebenberg et al., 2012; Maguire et al., 2014) or new programming tools (Bishop-Clark et al., 2007; Major et al., 2014) may be considered since these appear to increase enjoyment in learning programming.

The group-specific results showed a moderately strong positive relationship between enjoyment and programming grade in the novice programmers who obtained an excellent programming grade (β=*.268*), but a weak negative and not statistically significant relationship in the novice programmers who obtained a good programming grade (β=*-.100*).

8.3.1.2 Engagement constructs that did not predict programming grade

The higher-order emotional engagement construct did not predict programming grade. The relationship between emotional engagement and programming grade (β=*.019*) was weak and not significant. Specifically, the lower-order engagement constructs – *help-seeking, persistence*, *trial and error*, *interest* and *gratification* did not predict programming grade.

Help-seeking

The relationship between help-seeking and programming grade was initially hypothesised as a positive relationship. However, in this study, there was a weak negative but statistically significant relationship between help-seeking and programming grade (β=*-.149*). **Hypothesis (H12a):** Help-seeking in learning programming will have a positive effect on course grade is **NOT SUPPORTED**. The negative relationship between help-seeking and programming grade was not consistent with the findings of prior research. The published literature on learning theory found a positive relationship between help-seeking and the achievement of higher course grades (Bembenutty & White, 2013; Karabenick, 2003; Karabenick & Newman, 2006;), improved performance in a task (Nelson-Le Gall & Glor-Scheib, 1985), and Komarraju and Nadler (2014) confirmed the relationship between self-efficacy, help-seeking, and the achievement of grades.

The group-specific results may offer a plausible explanation for the overall negative relationship. The group-specific results showed a weak positive and not statistically significant relationship between help-seeking and programming grade in the novice programmers with an average programming grade (β=*.188*) compared to a weak negative

and not statistically significant relationship in the novice programmers with an excellent programming grade (β=-.*126*). However, since the results were not statistically significant, there is little credible evidence to suggest that help-seeking is not a predictor of programming grade and the finding is somewhat inconclusive.

Next, the recently published literature on learning theory and computer programming was examined for a plausible explanation for the overall negative findings. The higher acceptability to ask for help in programming and a lack of contextual help may be two plausible explanations for the negative relationship between help-seeking and programming grade.

In their study on the perception of students on plagiarism and programming, Aasheim, Rutner, Li, and Williams (2012) found that seeking help in programming assessments was more acceptable than seeking help on essay-related assessments. This finding could suggest that asking for help on programming assessments was fairly common and can be confirmed by the findings of the focus groups (Chapter 5, Section 5.3.2.2). Therefore, based on the overall negative findings in this study, it is possible that novice programmers who frequently ask for help are not likely to have an improved performance in their introductory programming course.

In yet another study, Li, Xing, Peng, and Zhao (2013) examined the help-seeking behaviour of a group of software developers and found that although help-seeking is important in software development, the "existing tools for finding and using help information are largely agnostic of the developers' working context" (p. 149). A similar explanation could be offered for the context of this study. Although novice programmers may be frequently seeking help with their programming assessments, it is possible that the help received was not useful for solving the errors in their program.

Persistence

The relationship between persistence and programming grade was initially hypothesised as a positive relationship. However, there was a weak negative and not statistically significant relationship between persistence and programming grade (β=-.*011*). **Hypothesis (H14a):** Persistence in learning programming will have a positive effect on course grade is **NOT SUPPORTED.** The negative relationship between persistence and programming grade contradicts the findings that were published in the literature on learning theory that persistence in learning leads to better performance (Glastra et al., 2004; White, 2004).

One plausible explanation for the negative relationship could be that the difficulties of learning programming (Chapter 2, Section 2.1.4.1) may have prevented the novice programmers from progressing in the course. This plausible explanation may be supported with evidence from prior research and from the focus group. In an experiment involving 10 programming students, Bennedsen and Caspersen (2012) found that the productivity of

programming students was hindered when the novice programmers encountered syntax related problems. This could suggest that despite persisting in resolving errors in programming assessments, the novice programmers may have experienced a loss of productivity, which then resulted in a negative effect on their programming grade.

Similarly, during the focus groups, the participants were asked to identify the attitudes that were important for achieving success in their introductory programming course. The participants identified *determination (JR, FG3; FA, FG4)*, *patience and attentiveness (MJE, FG4)*, *curiosity (CW, FG1; CL, FG2; MJE, FG4)*, *open-minded to every single possibility (MH, FG1)*, and *learning how to fix things (DI, FG2)* as persistence-related attitudes. While identifying persistence-related attitudes, the participants from the focus groups had not raised any concerns that might suggest a loss of productivity. This implies that the novice programmers felt that they should persist in their introductory programming course but were unaware that their persistence could lead to running out of time to complete their assessments, or that they had possibly focused on resolving programming related problems that did not have a significant contribution to their programming assessment.

The group-specific results showed a weak positive relationship between persistence and programming grade in the novice programmers who obtained an average programming grade (β=*.138*) compared to a weak negative relationship in the novice programmers with who obtained an excellent programming grade (β=*-.156*). However, these relationships were not statistically significant which implies that there is no credible evidence to support the differences in the strength of the relationships between the groups.

The second group-specific result revealed a weak positive and not statistically significant relationship between persistence and programming grade in the novice programmers whose programming self-efficacy beliefs had dropped by the end of their introductory programming course (β=*.149*) compared to a weak negative relationship in the novice programmers whose programming self-efficacy beliefs had increased by the end of their introductory programming course (β=*-.116*). Here again, these relationships were not statistically significant which implies that there is no credible evidence to support the differences in the strength of the relationships between the groups.

Trial and Error

There was a weak negative and not statistically significant relationship between trial and error and programming grade (β=*-.062*). **Hypothesis (H17a):** A trial and error strategy in learning programming will have a positive effect on course grade is **NOT SUPPORTED**. Although hypothesis H17a is not supported in this study, the findings from the focus groups and existing literature on computer programming clearly show that novice programmers use a trial and error strategy to debug the errors in their programming code.

Interest

The relationship between interest and programming grade was initially hypothesised as a positive relationship. However, in this study, a weak negative but statistically significant relationship was observed between interest and programming grade (*β=-.167*). **Hypothesis (H18a):** Interest in learning programming will have a positive effect on course grade is **NOT SUPPORTED**. The negative relationship between interest and programming grades contradicts the findings from prior research which found positive correlations between interest and programming performance (McKinney & Denton, 2004; Wiedenbeck et al., 2007). On the other hand, other studies have acknowledged the importance of interest and staying motivated in the course but have not examined the correlation between interest and course grades (Bye et al., 2007; Sheard et al., 2010).

One plausible explanation for the weak negative relationship could be due to the wording of the items that were used to measure interest. The items appear to measure both value- and affect-related interest. O'Keefe and Linnenbrink-Garcia (2014) described affect-related interest as the emotions of the participants when performing a task, value-related interest as the degree of importance placed on performing a task well, and examined the effect of affect-related interest and value-related interest on a word-forming problem. One finding of their study revealed that participants with high affect-related interest, but with low value-related interest had low performance. They explained that participants with high affect-related interest may have "enjoyed the process of solving the problem or tried to figure out new strategies, and that this was more important than performing well" (p. 77). In this study, items EEIN1 and EEIN4 rely on comparisons to other courses and may be measured as value- or affect-related interest depending on which type of interest was more dominant in the participant while item EEIN3 measures affect-related interest. The participants may have made the comparison to other courses (items EEIN1 and EEIN4) based on value- or affect-related interest, resulting in the weak negative relationship between interest and programming grade.

Further, the overall weak negative relationship could be explained by one group-specific result which revealed a weak positive and not statistically significant relationship between interest and programming grade in the New Zealand group (*β=.015*) compared to a weak negative relationship in the Malaysia group (*β=-.208*). In addition, the negative relationship was stronger in the novice programmers with a good intelligence score (*β=-.271*) compared to the novice programmers with an average intelligence score (*β=-.039*).

Gratification

The relationship between gratification and programming grade was initially hypothesised as a positive relationship. However, a weak negative and not statistically significant relationship was observed between gratification and programming grade (*β=-.068*). **Hypothesis (H21a):**

Gratification in learning programming will have a positive effect on course grade is **NOT SUPPORTED.** Since gratification is proposed as a new indicator of emotional engagement in this study, and no literature could be found to support the relationship between gratification and programming grade, a recommendation for further research is proposed in Chapter 10, Section 10.4 for the development of the gratification construct.

8.3.2 Self-assessment

In Chapter 3, Section 3.2.1, self-assessment was defined as "the individual's assessment of their abilities after they have completed a particular activity or task" (Mills et al., 2007, p. 421). The self-assessment construct was proposed to address the concern of relying on a single-item scale (programming grade) to measure the programming performance of novice programmers. Self-assessment was initially measured by 5 items but reduced to 3 items in the final set. Table 8.12 presents the final set of items that was used to measure self-assessment. The self-assessment construct satisfied all validity and reliability criteria (α = .759; CR = .862; AVE = .677). The engagement constructs moderately predicted 49.0% of the variance in the novice programmer's *self-assessment* of their programming performance (R^2 = *.490*). The items to measure the self-assessment construct were constructed from the definition of the construct.

Table 8.12: Items to measure self-assessment

Construct:	**Self-assessment**
Code	**Indicator**
SA1	I believe that I have performed well in this programming course.
SA2	I believe that I have learned adequate programming skills in this course.
SA5	I now understand the bigger picture of what programming is and what you can do with programming.

An extensive literature review was not conducted to support the findings of the relationship between the lower-order engagement constructs and self-assessment. This is because self-assessment was proposed as a second dependent variable that measured programming performance in order to compare if the engagement constructs that predicted programming grade had also predicted the novice programmer's self-assessment of their performance.

8.3.2.1 Engagement constructs that predicted self-assessment

All three of the higher-order engagement constructs predicted self-assessment. There was a moderately strong relationship between behavioural engagement and self-assessment (β=*.268*), between cognitive engagement and self-assessment (β=*.217*), and between emotional engagement and self-assessment (β=*.296*). Specifically, the lower-order engagement constructs that predicted self-assessment were similar to programming grade with the exception of gratification. The engagement constructs that predicted self-assessment was *effort*, *deep learning*, *surface learning*, *enjoyment*, and *gratification.*

Effort

There was a moderately strong positive and statistically significant relationship between effort and self-assessment (*β=.257*). **Hypothesis (13b):** Effort in learning programming will have a positive effect on self-assessment is **SUPPORTED**. When compared with the other lower-order engagement constructs, effort was ranked the second highest in its performance and importance in the IPMA (Chapter 7, Section 7.7). The positive relationship between effort and self-assessment was similar to the IPMA for effort in the dependent variable programming grade (Section 8.3.1.1).

Deep Learning

There was a weak positive but statistically significant relationship between deep learning and self-assessment (*β=.175*). **Hypothesis (15b):** A deep learning approach in learning programming will have a positive effect on self-assessment is **SUPPORTED.**

Surface Learning

There was a weak negative but statistically significant relationship between surface learning and self-assessment (*β=-.091*). **Hypothesis (16b):** A surface learning approach in learning programming will have a negative effect on self-assessment is **SUPPORTED**. When compared with the other lower-order engagement constructs, surface learning was ranked the lowest in its performance and importance in the IPMA (Chapter 7, Section 7.7). The negative relationship between surface learning and self-assessment was similar to the IPMA for surface learning in the dependent variable programming grade (Section 8.3.1.1).

The group-specific results revealed a negative relationship between surface learning and the participant's self-assessment of their performance in the New Zealand group (*β=-.188*) compared to a weak and not statistically significant relationship in the Malaysia group (*β=.000*).

Another group-specific finding revealed a weak positive and not statistically significant relationship between surface learning and the novice programmer's self-assessment of their performance in the novice programmers with an average programming grade (*β=.067*) compared to a negative and statistically significant relationship in the novice programmers with an excellent programming grade (*β=-.200*).

Enjoyment

There was a positive and statistically significant relationship between enjoyment and self-assessment (*β=.136*). **Hypothesis (19b):** Enjoyment in learning programming will have a positive effect on self-assessment is **SUPPORTED**.

Interestingly, when compared with the other engagement constructs, enjoyment was ranked the highest in its performance and importance in the IPMA (Chapter 7, Section 7.7). The positive relationship between enjoyment and self-assessment was similar to the IPMA for enjoyment in the dependent variable programming grade (Section 8.3.1.1).

Gratification

Unlike the negative relationship observed between gratification and programming grade, there was a positive but statistically significant relationship between gratification and self-assessment (*β=.136*). **Hypothesis (H21b):** Gratification in learning programming will have a positive effect on self-assessment is **SUPPORTED.**

The group-specific results revealed a weak positive and not statistically significant relationship between gratification and the participant's self-assessment of their performance in the novice programmers with an average (*β = .094*) or good programming grade (*β=.006*) compared to a strong positive relationship in the novice programmers with an excellent programming grade (*β=.292*). Interestingly, when compared with the other engagement constructs, gratification was ranked the highest in its performance and importance in the IPMA (Chapter 7, Section 7.7).

8.3.2.2 Engagement constructs that did not predict self-assessment

Although there was a moderate but statistically significant relationship between the higher-order engagement constructs and self-assessment, not all of the lower-order engagement constructs predicted self-assessment. Similar to the outcome of the programming grade dependent variable, the engagement constructs that did not predict self-assessment were *help-seeking*, *persistence*, *trial and error*, and *interest*.

Help-seeking

The relationship between help-seeking and self-assessment was initially hypothesised as a positive relationship. However, in this study, a weak negative and not statistically significant relationship was observed between help-seeking and self-assessment (*β=-.063*). **Hypothesis (H12b):** Help-seeking in learning programming will have a positive effect on self-assessment is **NOT SUPPORTED.**

Persistence

The relationship between persistence and self-assessment was initially hypothesised as a positive relationship. However, there was a weak positive and not statistically significant relationship between persistence and self-assessment (*β=.034*). **Hypothesis (H14b):** Persistence in learning programming will have a positive effect on self-assessment is **NOT SUPPORTED**.

Trial and Error

There was a weak positive and not statistically significant relationship between trial and error and self-assessment (*β=.047*). **Hypothesis (H17b):** A trial and error strategy in learning programming will have a positive effect on self-assessment is **NOT SUPPORTED**.

However, group-specific results revealed a weak negative and not statistically significant relationship between trial and error and the participant's self-assessment of their performance in the novice programmers with an average intelligence score (*β=-.127*) compared to a positive and not statistically significant relationship in the novice programmers with an excellent intelligence score (*β=.156*).

Interest

There was a weak positive and not statistically significant relationship between interest and self-assessment (*β=.088*). **Hypothesis (H18b):** Interest in learning programming will have a positive effect on self-assessment is **NOT SUPPORTED**.

8.4 Confounding Variables

Based on prior research, intelligence, and prior programming experience were proposed as two confounding variables in this study (Chapter 3, Section 3.3). After an extensive literature search, the novice programmer's school result was proposed as a measure for intelligence. Both the confounding variables did not significantly affect the dependent variables programming grade and self-assessment (Chapter 7, Section 7.8).

8.4.1 Intelligence

According to Alexander et al. (2003), pre-University results do not predict the novice programmer's success in programming. This study supports the finding by Alexander et al. (2003). The effect of intelligence, which is measured by school results, on the dependent variables *programming grade* (*β=.008)* and *self-assessment (β=.025*) was weak and not statistically significant. However, the weak and not statistically significant finding confirmed that the intelligence of a novice programmer was not a confounding variable in this study. As a result, there is a higher level of confidence that the dependent variables (programming grade and self-assessment) were influenced by the independent variables (programming self-efficacy and engagement) in this study.

8.4.2 Prior programming experience

The effect of prior programming experience on the dependent variables *programming grade* (*β=.004)* and *self-assessment (β=-.011*) was weak and not statistically significant. This finding confirms the finding by Ventura (2005) and Wilson and Shrock (2001) that prior programming experience does not influence programming success.

By contrast, other researchers have found that prior programming experience is linked to programming success (Alvarado, Lee, & Gillespie, 2014; Rountree et al., 2002; Watson, Li, & Godwin, 2014; Wiedenbeck, 2005).

Only 33.3% of the participants in this study had prior programming experience (Chapter 7, Section 7.2). Of these, only 9.7% of the participants with prior programming experience had rated that they had a higher than average programming experience. Thus, the small sample size could have resulted in a not statistically significant finding. In addition, since the number of novice programmers with prior programming experience was small, it was not statistically possible to perform a multi-group analysis based on the various levels of programming experience. However, the weak and not statistically significant finding confirmed that prior programming experience was not a confounding variable in this study. As a result, there is a higher level of confidence that the dependent variables (programming grade and self-assessment) were influenced by the independent variables (programming self-efficacy and engagement) in this study.

8.5 The final research model for self-efficacy, engagement, and programming performance

Figure 8.1 presents the final structural model in this study. The structural model shows the relationships between self-efficacy, the lower-order engagement constructs, and programming performance. The strong positive relationship between pre-programming self-efficacy and post-programming self-efficacy confirmed that the self-efficacy beliefs of novice programmers had increased as they progressed in their introductory programming course. The programming self-efficacy beliefs of the novice programmers then influenced their engagement behaviour during their introductory programming course, and the relationships are depicted by the arrows that connect the post-programming self-efficacy construct to the lower-order engagement constructs in Figure 8.1. The engagement of the novice programmer then predicted their programming performance, and the relationships are depicted by the arrows that connect each indicator of engagement to the programming performance (measured using programming grade and self-assessment) of the novice programmer. The grey arrows represent relationships that are not supported.

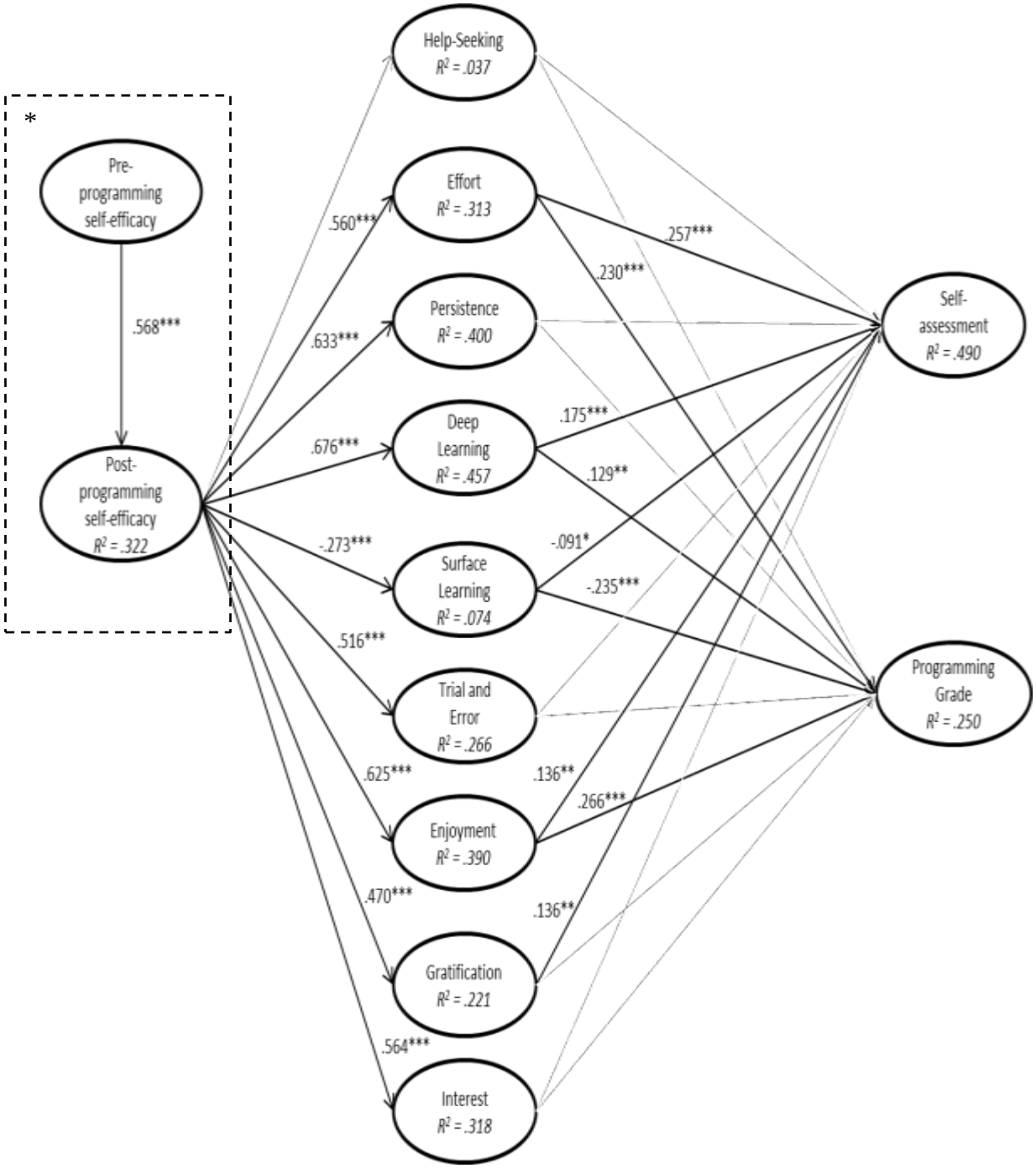


Figure 8.1: The final structural model for self-efficacy, lower-order engagement constructs, and programming performance.

***In Chapter 9, Section 9.3.1, the predictability of the relationships between programming self-efficacy and the engagement factors are discussed and arguments are put forth to support the claim that post-programming self-efficacy influences engagement.*

Since engagement is a multi-dimensional construct, the indicators of engagement are presented as higher-order engagement constructs in Figure 8.2. All the relationships in the higher-order engagement model were supported with the exception of the relationship between emotional engagement and programming grade.

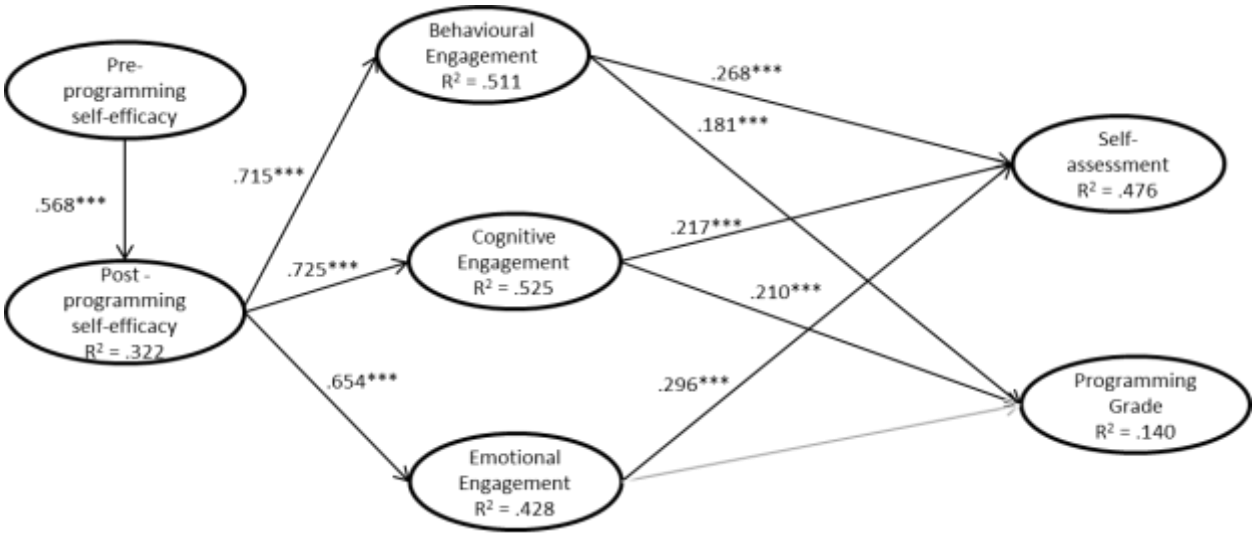


Figure 8.2: Final structural model for self-efficacy, higher-order engagement constructs, and programming performance

8.6 Chapter Summary

This study found that programming self-efficacy influenced the engagement behaviour of the novice programmers in their introductory programming course, with the exception of help-seeking, which is an indicator of behavioural engagement. These findings answered Research Question 1 (RQ1). Contrary to the findings from existing published research on learning theory, this study found a negative relationship between programming self-efficacy and help-seeking. The findings from the multi-group analysis by country and programming grade offered plausible explanations for the overall negative finding.

This study then found that effort, enjoyment, surface learning, and deep learning predicts programming performance. Effort, enjoyment, and deep learning had a positive effect on programming performance while surface learning had a negative effect on programming performance. These findings answered Research Question 2 (RQ2). Plausible explanations were also offered for the relationships that did not predict programming performance by drawing upon the group-specific results in this study and from existing published literature on learning theory and computer programming.

The effect of the confounding variables on the programming performance of the novice programmers was not significant. The final research model for this study is presented in Figure 8.1 while Figure 8.2 shows the research model when the indicators of engagement are presented as higher-order engagement constructs.

Chapter 9: Further Discussion

The discussion in Chapter 8 summarised the results of this study, confirmed the hypotheses, and interpreted the results by offering plausible explanations for hypotheses that was not supported. This chapter offers further insight into the findings of this research by examining the extent in which the results may be generalised to the sample population, the impact of intelligence as a confounding variable, the predictability claims in this study, the engagement factors, gender-based differences, and the validation of programming grade. This thesis argues that the key findings have implications for course instructors and proceeds to make recommendations about the design and delivery of introductory programming courses.

9.1 Generalisability of the findings

One benefit of survey research is the findings may be generalised to the larger population of novice programmers (Pinsonneault & Kraemer, 1993; Punch, 2005). However, the extent to which the findings may be generalised to the larger population should also be considered. Heterogeneity, gender imbalance, uneven grade distribution, and the lack of information on the attrition rate were considered to determine the extent of the generalizability of this study.

This study benefits from a fairly large heterogeneous group of participants (433 participants). The data was collected in two countries that had students from various ethnicities enrolled in the course. In each country, data was collected from several Higher Educational Institutions (HEIs), and the programming languages that were used in the introductory programming courses varied between the HEIs. By contrast, the number of participants in related studies (Chapter 2, Section 2.1.4.2) that had examined the behavioural factors of novice programmers in introductory programming courses were less than 150 participants with the exception of the study by Askar and Davenport (2009) which comprised of 326 participants, and the study by Ramalingam and Wiedenbeck (1998) which comprised of 421 participants. Additionally, these related studies did not report the number of students that were invited to participate in their study while in this study as many as 1093 novice programmers from several introductory programming courses were invited to participate in this study.

Another aspect that contributes to the heterogeneity of the participants in this study relates to the data collection in two countries - Malaysia and New Zealand, both of which have a mix of local and international students of various ethnicities (Chapter 7, Section 7.2), and in each country the data was collected at several Higher Educational Institutions (HEIs). The number of participants from each country was almost equal, with 49.4% of the participants from HEIs in New Zealand, and 50.6% of the participants from HEIs in Malaysia. This allowed for a balanced perspective of the behaviour of novice programmers in introductory programming courses to be obtained in both countries. Each of the HEIs had also used different programming languages in their introductory programming course. By contrast, the participants in prior studies were homogeneous. The participants were from one HEI and had

learned the same programming language in their introductory programming course. As a result of the heterogeneity of this study, there is a higher level of confidence in generalising the findings of this study to the larger population of novice programmers in introductory programming courses.

However, the generalisation of the findings may not apply to all novice programmers. The number of female participants in this study was small. The percentage of male participants was 72.7% while females only made up 27.3% of the participants in this study (Chapter 7, Table 7.2). The gender imbalance in this study could imply that the findings of this study may be more generalised to male students in introductory programming courses compared to female students. However, when the data was compared by gender (Chapter 7, Section 7.11.5), only two relationships showed a slight but statistically significant difference in the strength of their effect on male and female novice programmers. These relationships are between *post-programming self-efficacy* and *help-seeking*, and between *post-programming self-efficacy* and *deep learning*. Thus, despite the gender imbalance, the findings of this study suggest that the results may be generalised to the male and female novice programmers, and a plausible explanation may be offered for the differences in the strength of the relationships between *post-programming self-efficacy* and *help-seeking*, and between *post-programming self-efficacy* and *deep learning* between male and female novice programmers. These relationships are discussed further in Section 9.4.3 and Section 9.5.

An uneven grade distribution was another factor that was considered when determining the extent of the generalizability of this study. The number of participants who performed poorly in their introductory programming course was small. Only 8.8% of the participants in this study obtained a Grade D or lower while 68.6% of the participants obtained, at least, a grade B in their introductory programming course (Chapter 7, Table 7.2). Generalising the findings of this study may not apply to this group of novice programmers due to the small number of participants who obtained a Grade D or lower. Self-selection bias is one factor which may have led to the small number of participants in the group that received a grade D or lower. It may be possible that the novice programmers that had low self-efficacy beliefs in succeeding in their introductory programming course and that had not been performing well had chosen not to participate in this study. Thus, the findings of this study may be more generalised to novice programmers who obtained a good (Grade B) or excellent (Grade A) programming grade in their introductory programming course.

Attrition rate data was the next factor that was considered when determining the extent of the generalizability of this study. Attrition rate data could not be obtained from the participating HEIs due to institutional policies. Additionally, since participation was voluntary in this study, it may be possible that the 38.7% of the participants that had dropped out by Phase 3 of this study (Chapter 7, Section 7.2), had not been performing well in the course, had significantly lower self-efficacy beliefs and had chosen not to complete the survey or had dropped out of the introductory programming course.

Despite the lack of participation of novice programmers who had performed poorly in the course and the lack of attrition rate data, this study benefits from a large number of participants that had obtained a higher than average to excellent results in their introductory programming course. This implies that the result of this study is able to reliably show the behaviour of novice programmers that had performed very well in their introductory programming course, which the weaker performing novice programmers can benefit from.

9.2 Intelligence as a confounding variable

Although intelligence was proposed as a confounding variable to the dependent variable - programming performance and a not statistically significant relationship was established, a multi-group analysis was nevertheless performed to determine if there were any significant differences among novice programmers with average, good, or excellent intelligence scores. The group-specific results revealed several differences and the results were reported in Chapter 7, Section 7.11.2, and were discussed in Chapter 8.

This study found that the programming self-efficacy beliefs of novice programmers with an average intelligence score had increased significantly during the introductory programming course compared to the novice programmers with a good or excellent intelligence score (Chapter 8, Section 8.2). There was also a notable increase in the strength of the relationship between post-programming self-efficacy and the engagement of novice programmers with an average intelligence score. Specifically, the novice programmers with an average intelligence score had significantly stronger relationships compared to novice programmers with a good intelligence score in the relationships between post-programming self-efficacy and enjoyment, between post-programming self-efficacy and persistence, and between post-programming self-efficacy and trial and error. These findings appear to be novel and have not been reported in prior research.

One plausible explanation for these findings could be that novice programmers with an average intelligence score entered the introductory programming course with lower self-efficacy beliefs due to their mediocre performance in school and that their self-efficacy beliefs had increased significantly as they progressed in their course compared to their peers who had a good or excellent intelligence score. Subsequently, the increase in their self-efficacy beliefs resulted in significantly higher levels of enjoyment, persistence, and the use of a trial and error strategy in their introductory programming course compared to novice programmers with a good or excellent intelligence score.

By contrast, the strength of the relationship between post-programming self-efficacy and the engagement of novice programmers with a good intelligence score were lower than the novice programmers with an average intelligence score while the relationship between post-programming self-efficacy and the engagement of novice programmers with an excellent intelligence score were similar to the novice programmers with an average intelligence score.

Clearly, the varying strength of the relationships among the three groups of novice programmers was due to their confidence levels based on their past performance. Bandura's (1977) Social Cognitive Theory proposed 4 sources by which individuals may develop their self-efficacy beliefs, and these are discussed in Chapter 2, Section 2.2.3. Performance accomplishment is one source of self-efficacy belief that appears to explain the comparatively lower increase in the programming self-efficacy beliefs of the novice programmers with a good or excellent intelligence score. The novice programmers with a good or excellent intelligence score had entered the course with strong self-efficacy beliefs since they performed very well in school, and had subsequently expected to be successful in their introductory programming course, resulting in a lower increase in their self-efficacy beliefs compared to their peers with an average intelligence score.

Interestingly, the relationship between the engagement factors and programming performance showed little significant difference among the novice programmers with average, good, and excellent intelligence. Thus, the higher increase in the self-efficacy beliefs of the novice programmers during the introductory programming course and their increased enjoyment, persistence, and use of a trial and error strategy did not lead to a significantly higher performance compared to the novice programmers with a good or excellent intelligence score.

9.3 Predictability claims

Although the data in Phase 1 and Phase 3 of this research was collected using a cross-sectional survey, the data collection at three intervals suggests a longitudinal study which has implications for the predictability claims in this research. The pre-programming self-efficacy data was collected at the beginning of the introductory programming course while the post-programming self-efficacy data, the engagement data, and the participant's self-assessment of their performance data was collected at the end of the introductory programming course. The programming grade data was obtained between 6 and 10 weeks after the novice programmers completed their assessments. This would typically be between 8 and 12 weeks after the end of the introductory programming course.

9.3.1 Predictability of the relationship between post-programming self-efficacy and engagement

According to Gregor and Klein (2014), the claims for causal relationships should have time precedence and other explanations for the cause and effect relationship must be eliminated. In the latter requirement, two potential confounding variables were identified and found to have no significance on the dependent variables in this study (Chapter 8, Section 8.4). However, in terms of time precedence, the data collection at three intervals may allow for a predictive claim to be made between post-programming self-efficacy and engagement although the data for these two constructs were collected at the same time. This is because the statistical tests showed that there is a significant relationship between pre-programming

self-efficacy and engagement and the results are presented in Appendix J. The pre-programming self-efficacy data and engagement data were collected at two-time intervals which satisfy the time precedence that was suggested by Gregor and Klein (2014). However, the strength of the relationships between pre-programming self-efficacy and the engagement factors was lower than the relationships between post-programming self-efficacy and the engagement factors. The comparatively lower strength in the relationships could be due to the pre-programming self-efficacy data being collected at the beginning of the introductory programming course when novice programmers are introduced to the course.

In addition, there was a significant increase in the programming self-efficacy beliefs of the novice programmers as they progressed in their introductory programming course. As a result, the claim that post-programming self-efficacy influences the engagement of novice programmers is made since the research model is able to reliably show that the programming self-efficacy of novice programmers had increased over time and had strengthened the relationship between post-programming self-efficacy and the engagement of the novice programmers.

9.3.2 Predictability of the relationship between engagement and programming grade

The data for the engagement factors and programming grade was collected at two-time intervals which satisfy the requirement for predictability claims by Gregor and Klein (2014). In addition, two potential confounding variables were identified and did not have a significant effect on the dependent variable programming grade (Chapter 8, Section 8.4), which satisfy the second requirement that was outlined by Gregor and Klein (2014) regarding claims of causality. Therefore, this thesis is able to claim a predictive relationship between the engagement factors and the programming performance of the novice programmer.

9.3.3 Predictability of the model in this study

Overall, since there appears to be no related work conducted that is of a similar magnitude and scale to this study, the influence of programming self-efficacy on the engagement of novice programmers, and the predictability of the relationships between the engagement factors and the programming performance of the novice programmer could not be compared in its entirety. Nevertheless, this study provides a reliable predictive model because this model benefits from a heterogeneous group of participants compared to related work that had only examined some of the relationships that were hypothesised in this study. The related work is discussed in Chapter 2, Section 2.1.4.2, and the heterogeneity of the participants in this study is discussed in Section 9.1.

9.4 Examination of engagement constructs in this study

This section offers further insight into the engagement constructs that were examined in this research. The insights include interpretations of the expected and unexpected hypotheses in this study, issues in the conceptualisation and development of the constructs, and the impact of the group-specific results that were briefly discussed in Chapter 8.

9.4.1 Participation

The participation construct was dropped during the scale development stage due to a lack of reliable measures. This does not imply that participation is not an important engagement factor in introductory programming courses. Instead, participation may be a reliable measure when only a single cohort of introductory programming course is studied, or the same method of instruction, institutional policies, and assessment methods are applied to all cohorts of introductory programming courses. This is because the interpretation of what is participation may differ from one introductory programming course to another and from one novice programmer to another.

The frequency of attendance at lectures and practical sessions (item BEPA2) is one measure that was proposed and found unreliable during the scale development stage. This measure may be appropriate in introductory programming courses that make attendance mandatory. However, the mandatory attendance at lecture and practical sessions could not be established since this study was conducted in several HEIs that may have different policies on attendance at lecture and practical sessions. The same issue applies to item BEPA3 which measures discussion with friends and family members. Here again, an assumption should not be made that novice programmers should be actively engaged in discussions and that for those who do not, they are assumed as not participating in the course. Like the measure for attendance, this measure may be appropriate in introductory programming courses that make discussions mandatory or are assessed.

In addition, as pointed out by one expert reviewer (Chapter 6, Section 6.2.1.4), the physical presence of a student at their lecture does not imply that the student is engaged with the lecture. On further reflection, evidence from the focus groups concurs with the views of the expert reviewer, in that each novice programmer participates differently in the course (Chapter 5, Section 5.3.2). Participant RB in Focus Group 3 explained that she attends lectures so that she does not feel guilty, and continued to explain that she tended to "zone out" during lectures. Participant CW from Focus Group 1 explained that he felt obliged to attend lectures although he did not see much value in attending the lectures. These observations raise the issue of distinguishing between active and passive participation. Novice programmers who actively participate in the course by asking questions during lectures and practical sessions may be assumed to be participating in the course while novice programmers who prefer to work on their own and seek answers to programming problems by referring to online resources and reference books may be assumed to be passively participating in their introductory programming course. Due to this, the participation construct should be clearly conceptualised by establishing the method of instruction, institutional policies, and assessment methods so that the items that are developed to measure participation in the course are reliable.

9.4.2 Effort

Effort had a strong positive and statistically significant relationship with programming grade and was ranked the second highest in its performance and importance in the Importance Performance-Matrix Analysis (IPMA) (Chapter 8, Section 8.3.1.1). This finding suggests that effort is an important predictor of success in introductory programming courses.

However, one surprising finding in the multi-group analysis showed a weak negative and not statistically significant relationship between effort and programming grade in the Malaysia group of novice programmers, but a strong positive and statistically significant relationship in the New Zealand group of novice programmers (Chapter 8, Section 8.3.1.1). One explanation for this surprising finding lies in the distribution of the programming grade in the two countries that were sampled in this study (Chapter 7, Table 7.2). A larger percentage of novice programmers from the New Zealand group had obtained an excellent programming grade (Grade A). By contrast, a larger percentage of novice programmers from the Malaysia group had obtained an average programming grade (Grade C, D or E). This seemingly skewed grade distribution does not imply that the novice programmers in New Zealand were performing better than the novice programmers from HEIs in Malaysia. Instead, self-selection bias may be one factor that explains the difference in the distribution of the programming grades between Malaysia and New Zealand and is discussed in Section 9.1.

Yet another explanation for the surprising finding in the multi-group analysis between the countries that were sampled in this study is the finding that shows a strong positive relationship between effort and programming grade in the novice programmers with an excellent programming grade compared to a weak negative and not statistically significant relationship in the novice programmers with an average or good programming grade (Chapter 8, Section 8.3.1.1). When the participants in this study were grouped by programming grade and country, a large percentage (72.2%) of the 162 participants that had obtained an excellent programming grade were from HEIs in New Zealand while the remaining 27.8% were from HEIs in Malaysia. Additionally, out of the 136 participants that had obtained an average programming grade, a large percentage (72.1%) were from HEIs in Malaysia while the remaining 27.9% were from HEIs in New Zealand (see Chapter 7, Table 7.2 for distribution of programming grade). On the other hand, the percentage of novice programmers from the New Zealand group and from the Malaysia group that had obtained an average programming grade (Grade B) was somewhat balanced with 43.7% in the former group (New Zealand) and 56.3% in the latter group (Malaysia).

Therefore, the weak negative and not statistically significant relationship between effort and programming grade in the Malaysia group was due to the majority of novice programmers in the Malaysia group obtaining an average (44.8%) or good (34.7%) programming grade. Overall, there was a weak negative and not statistically significant relationship between effort and programming grade in the novice programmers with an average and good programming grade. Since the negative relationship was not statistically significant, there is

little credible evidence to suggest that effort is not a predictor of success in the novice programmers who had obtained an average or a good programming grade.

Instead, it would seem that effort is a strong predictor of success in at least the group of novice programmers with an excellent programming grade, since the majority of the novice programmers in the New Zealand group had obtained an excellent programming grade (54.7%). Overall, there was a strong positive relationship between effort and programming grade in the novice programmers with an excellent programming grade.

The strong positive relationship between effort and programming grade in the novice programmers with an excellent programming grade suggests that novice programmers who did not perform as well can learn from the excellent performers that had expended effort to learn programming leading to excellent programming grades. Thus, based on the evidence from the focus groups and items that were used to measure effort in this research, three suggestions may be put forward for expanding effort in order to be successful in learning programming. The first suggestion is to consistently work on the programming assessment throughout the course by staying on top of the programming assessments. This includes starting the assessment early and completing the assessment well before the deadline. Secondly, practise appears to be another factor that shows that novice programmers are expending effort in learning programming. By practising writing programs, the novice programmer can strengthen their understanding of the programming concepts. Finally, setting aside time to work on programming assessments and effective time management are two strategies that appear to enable novice programmers to expend effort when learning programming.

9.4.3 Help-seeking

The overall negative relationship between post-programming self-efficacy and help-seeking, and between help-seeking and programming grade was unexpected. The findings from recently published literature on learning theory and computer programming were able to shed some light on the negative relationship and are discussed in Chapter 8, Section 8.3.1.2.

One further reflection, another factor that may have contributed to the overall negative relationship could be the lack of clarity in the wording of items BEHS1 and BEHS3. The wording of these items may not have clearly distinguished executive help-seeking strategy from instrumental help-seeking strategy. This is because, unlike item BEHS5, items BEHS1 and BEHS3 did not clearly state the novice programmer's intention to seek help which could determine if the item measures instrumental help-seeking or executive help-seeking. Help-seeking strategies are discussed in Chapter 3, Section 3.2.3.2, whereby students who use an instrumental help-seeking strategy expect hints and not an answer to solve the problem. Conversely, students who use an executive help-seeking strategy want to avoid work and save time.

The multi-group analysis may offer another perspective to the overall negative relationship. There was a moderately strong positive relationship between post-programming self-efficacy and the need to seek help in the Malaysia group compared to a moderately strong negative relationship in the New Zealand group (Chapter 8, Section 8.2.1.1). Like the effort construct in Section 9.4.2, a similar explanation could be offered for the differences between the Malaysia and New Zealand group. The positive relationship in the Malaysia group was due to the significantly higher percentage of novice programmers who had obtained an average programming grade in the Malaysia group (44.8%) compared to only 17.7% of novice programmers in the New Zealand group who had obtained an average programming grade. When the participants were grouped by programming grade, the average performers had a moderately strong positive and statistically significant relationship between post-programming self-efficacy and the need to seek help. By contrast, the novice programmers with a good programming grade had a moderately strong negative and statistically significant relationship between post-programming self-efficacy and the need to seek help while the excellent performers had a weak negative and not statistically significant relationship between post-programming self-efficacy and the need to seek help.

On further reflection, inadequate preparation and different learning preference could be two other factors that may explain the negative relationship between help-seeking and programming grade. Novice programmers who were facing difficulties when learning programming and were not well prepared for their programming assessments may have frequently asked for help by asking for solutions to the programming problems. However, in doing so, the novice programmer may not have thoroughly understood the programming problem and the solution that was applied to the problem which may have led to a lower programming grade. This factor also links to the earlier observation that the items to measure help-seeking did not clearly distinguish between instrumental and executive help-seeking strategies.

Different learning preference is another factor that may have led to the negative relationship between help-seeking and programming grade and links to one observation that was raised in the development of the participation construct (Section 9.4.1). Some participants in this study may have preferred to work on their own and seek answers to programming problems by referring to online resources and reference books instead of asking for help from other individuals such as their peers, tutors, course instructors, or family members. However, the items to measure help-seeking were mainly focused on seeking-help from other individuals and were less adequate for novice programmers who sought help from other resources.

9.4.4 Persistence

Section 9.2 provides further insight into the finding that novice programmers with an average intelligence score had a significantly stronger positive relationship between post-programming self-efficacy and persistence compared to the novice programmers with a good intelligence score (Chapter 8, Section 8.2.1.3). On the other hand, this study found that

persistence when learning to program does not necessarily lead to programming success. There was a negative but not statistically significant relationship between persistence and programming grade. The recent literature on computer programming cited loss of productivity as a plausible explanation for the negative relationship between persistence and programming performance (Chapter 8, Section 8.3.1.2). Although the group-specific results were not statistically significant, the novice programmers that had an increase in their self-efficacy beliefs during their introductory programming course, and novice programmers who obtained an excellent programming grade showed a negative relationship between persistence and programming grade. This negative finding suggests that novice programmers who excel in programming might find their assessments easier and require less persistence. This is because novice programmers who excel in programming will have a solid understanding of programming concepts, may be able to program with fewer errors, and may be able to debug their programming errors without much difficulty.

9.4.5 Deep Learning and Surface Learning

During the development of the conceptual research model (Chapter 3, Section 3.2), deep learning and surface learning are two of three learning strategies that were proposed to measure the cognitive engagement of novice programmers in introductory programming courses. While there was a clear positive relationship between post-programming self-efficacy and the use of a deep learning approach, and between the use of a deep learning approach and the programming performance of the novice programmer, the relationships between post-programming self-efficacy, surface learning, and programming performance showed that novice programmers who obtained an average programming grade tended to use a surface learning approach.

There was a weak negative but not statistically significant relationship between post-programming self-efficacy and surface learning in the Malaysia group compared to a moderately strong negative relationship in the New Zealand group (Chapter 8, Section 8.2.2.2). While it is clear that the New Zealand group refrained from using a surface learning approach to learning programming as their self-efficacy beliefs increased, the same interpretation could not be reached for the Malaysia group. Instead, it would seem that the Malaysia group had continued to use a surface learning approach although their self-efficacy beliefs increased during their course.

However when the results of this study were examined further, like the effort construct in Section 9.4.2, a similar explanation could be offered for the differences between the Malaysia and New Zealand group. The weak negative but not statistically significant relationship between post-programming self-efficacy and surface learning in the Malaysia group was due to the significantly higher percentage of novice programmers who had obtained an average programming grade in the Malaysia group (44.8%) compared to only 17.7% of novice programmers in the New Zealand group who had obtained an average programming grade. In addition, the group-specific result clearly showed a weak positive but not statistically

significant relationship in the novice programmers who obtained an average programming grade (Chapter 8, Section 8.2.2.2). Clearly, the weak relationship shows that the novice programmers with an average programming grade had used a surface learning approach although their self-efficacy beliefs had increased and may even be unaware that a surface learning approach was not effective for learning programming.

The same interpretation may also be offered for the relationship between surface learning and self-assessment when compared by country and programming grade (Chapter 8, Section 8.3.2.1). The results show that the novice programmers who obtained an average programming grade had a weak positive but not statistically significant relationship between surface learning and self-assessment. This result, and the finding that there were no significant differences in the relationship between surface learning and programming grade between the groups suggests that the novice programmers who obtained an average programming grade had felt that their surface learning approach could lead to success in their introductory programming course and were unaware that a surface learning approach was not an effective strategy for learning programming.

9.4.6 Trial and Error

Trial and error was proposed as an indicator of cognitive engagement based on findings from prior research on learning programming that suggested that novice programmers used a trial and error strategy to learn programming. Further, evidence from the focus groups in this study confirmed the findings from prior research that novice programmers tended to use a trial and error strategy to learn programming (Chapter 5, Section 5.3.3.3). Despite these findings, a strong positive relationship between trial and error and programming performance could not be established in this study, although there was a strong positive relationship between post-programming self-efficacy and trial and error and the reasons for this were established in Chapter 8, Section 8.2.2.3. On further reflection, the measures for trial and error and poor debugging skills are two plausible explanations for the weak results.

The first plausible explanation lies in the items that were used to measure trial and error. Edwards (2004) argued that although a trial and error strategy may be sufficient for novice programmers to debug the errors in their programming code, adopting a reflective approach when learning programming may lead to better performance. Edwards' argument suggests that novice programmers who know how to apply the programming concepts that they learned, and had tried different solutions to solve their programming problems, may have used the trial and error strategy effectively, leading to better performance. His argument led to the re-examination of the items to measure trial and error (Chapter 8, Section 8.2.2.3, Table 8.8) in order to determine if the wording of the items distinguishes between effective and ineffective trial and error strategies when solving programming problems, and in particular when debugging programming errors. Items CETE2 and CETE3 suggest that novice programmers may have used an effective trial and error strategy as they were able to apply several possible solutions to solve their programming problems and had selected the

strategy that led to the best outcome. By contrast, items CETE5 and CESL4 suggest that novice programmers may have used an ineffective trial and error strategy as they had moved their codes around or inserted codes which they felt were correct to them without understanding the concepts that they had learned in their introductory programming course.

The second plausible explanation for the weak relationship between trial and error and programming performance could be due to the poor debugging skills, particularly in the novice programmers with an average intelligence score. During the focus groups, the participants explained that they often did not know how to solve their programming errors, and had used a trial and error strategy in an attempt to identify and solve their programming errors (Chapter 5, Section 5.3.3.3). This observation and the finding that there was a weak and not statistically significant negative relationship between trial and error and programming grade led to concerns that novice programmers may have poor debugging skills.

A literature search on debugging tools confirmed that novice programmers found debugging a challenge and that several researchers had attempted to develop debugging tools such as HelpMeOut (Hartmann, MacDougall, Brandt, & Klemmer, 2010) and Backstop (Murphy, Kim, Kaiser, & Cannon, 2008) that were intended to help novice programmers interpret and provide useful solutions for their programming errors. Although these debugging tools have alleviated some of the challenges that the novice programmers face when debugging their programming errors, being reliant on the tools to debug their programming errors may add to the steep learning curve in their introductory programming course. Further, since these debugging tools are not used in the software industry, the novice programmers are then faced with the difficulty of transitioning to using the debugging tool that is embedded in the programming environment (Integrated Development Environment - IDE).

The introductory programming courses of the participating HEIs were then examined to determine if the course outlines included exposing the novice programmers to strategies for debugging their programs. Not surprisingly, the course outlines indicated that the main objective of the introductory programming course was to introduce and examine fundamental programming concepts. But when the week by week plan of the introductory programming courses were examined, there was little to no emphasis on interpreting programming errors and the strategies that may be used to debug programming errors.

On the other hand, evidence from the large body of research published in Computer Science-related journals shows that the software industry places heavy emphasis on testing software applications and programmers typically use debugging tools and strategies to interpret and resolve their programming errors. The debugging tool that is embedded in the IDE is frequently used by software developers to step through the errors in the program line by line. In addition, software developers typically use a set of guidelines to debug the errors in their computer program, and frequently perform a set of tests to ensure that the computer program

functions without any errors. This practice is clearly not emulated in the introductory programming courses that were sampled in this study which could have resulted in the novice programmers using a trial and error strategy to debug their programming errors.

9.4.7 Enjoyment

Enjoyment had a strong positive and statistically significant relationship with programming grade and was ranked the highest in its performance and importance in the Importance Performance-Matrix Analysis (IPMA) (Chapter 8, Section 8.3.1.1). This finding suggests that enjoyment is an important predictor of success in introductory programming courses. With the exception of the multi-group analysis by programming grade, the relationship between post-programming self-efficacy, enjoyment and programming performance did not show any significant differences between the groups. There was a moderately strong positive relationship between enjoyment and programming grade in the novice programmers who obtained an excellent programming grade. However, the findings were not statistically significant in the novice programmers who obtained an average or good programming grade (Chapter 8, Section 8.3.1.1). Clearly, enjoyment is an important engagement factor for success in programming in at least the novice programmers who obtained an excellent programming grade. The findings were less obvious in the novice programmers who obtained an average or good programming grade and this implies that the novice programmers who obtained an average or good programming grade in their introductory programming course had not enjoyed learning programming as much as their peers who obtained an excellent programming grade, but had nevertheless worked through the course to meet the mandatory requirements of passing their introductory programming course.

9.4.8 Gratification

This study found that gratification has the potential to be an indicator of emotional engagement in introductory programming courses. There was a strong positive relationship between post-programming self-efficacy and gratification, and a positive relationship between gratification and self-assessment, particularly in the novice programmers who obtained an excellent programming grade. However, a similar finding could not be observed in the relationship between gratification and programming grade since the weak negative results was not statistically significant. These findings show a disconnect between the novice programmer's judgment of their performance and the achievement of a better programming grade when they felt gratified upon seeing the output of their program. One plausible explanation for the disconnect could be that although the novice programmers were able to see the output of their program and make their program work without errors, they may not have met all the requirements of the programming assessment that would lead to a better programming grade.

9.4.9 Interest

The overall weak negative relationship between interest and programming grade was unexpected and a plausible explanation was offered in the discussion in Chapter 8, Section 8.3.1.2. The plausible explanation relates to the lack of distinction between value- and affect-related interests in the items that measure interest. On further examination of the multi-group analysis, no further insights could be offered for the overall weak negative relationship.

9.5 Gender-based differences

By and large, there were no significant differences in the relationships between post-programming self-efficacy and engagement when the participants were grouped by gender. The absence of a significant difference in most of the relationships between post-programming self-efficacy and engagement suggests that the female participants in this study demonstrated self-efficacy beliefs, and engagement behaviour that was similar to the male participants and this finding is consistent with the findings by Ramalingam & Wiedenbeck (1998). However, there was a significantly stronger negative relationship between post-programming self-efficacy and help-seeking (Chapter 8, Section 8.2.1.1), and a significantly lower positive relationship between post-programming self-efficacy and deep learning (Chapter 8, Section 8.2.2.1) in the female participants. One factor that could explain the difference in the strength of the relationships between the genders could be that female novice programmers tend to under-report their confidence and engagement level, although this explanation does not hold true for the rest of the relationships that were examined in this study.

Interestingly, in contrast to prior research which found that female novice programmers had significantly lower self-efficacy beliefs and expectations of success in their programming course (Doubé & Lang, 2012), and a lower achievement of their learning outcomes (Rubio et al., 2015), this study did not find any significant differences in the relationships between the engagement factors and programming performance when the participants were grouped by gender. The absence of a significant difference in the relationships between the genders could suggest that the female participants in this study demonstrated engagement behaviour that led to programming success similar to the male participants. On the other hand, the absence of a significant difference in the relationships between the genders may also be due to self-selection bias. The small percentage of female novice programmers (27.3%) that had participated in this study may be made up of those who strongly felt that they would succeed in the course and were deeply engaged in the course compared to other female novice programmers who may have chosen not to participate in this study.

9.6 Validating Programming Grade

In Chapter 3, Section 3.2.1, after a detailed literature review of the measures for learning and success in programming, the programming grade of the novice programmer was proposed as a dependent variable in this study. Using programming grade as a measure of success in

introductory programming courses has its weaknesses. Firstly, programming grade may not reflect the true programming ability of the novice programmer, and secondly, the assessment design in some introductory programming courses may include non-programming related tasks or activities. In order to mitigate the weaknesses, and to validate the use of programming grade to measure programming performance, the course outline was obtained from the instructors of each introductory programming course in order to examine the method of assessment and the learning outcomes in the course.

There were between two and three assessments in each introductory programming course with the exception of two introductory programming courses that had five assessments. The assessments were largely made up of programming assignments, examinations, and/or tests while two introductory programming courses in New Zealand partially assessed the performance of the novice programmers using lab-based exercises that contributed to no more than 30% of the overall programming grade. By and large, the programming assignments were attempted individually with the exception of two introductory programming courses in Malaysia that required the assignment to be completed in a group although no more than 35% of the overall programming grade was from the group assignment. The breakdown of marks between the assignments, and examinations and/or tests in the introductory programming courses in Malaysia was somewhat consistent. The assignments normally contributed between 30% and 50% to the overall programming grade. However, there was a larger disparity in the breakdown of marks between the assignments and/or lab-based exercises, and examinations and/or tests in the introductory programming courses in New Zealand. The New Zealand assignments and/or lab-based exercises contributed between 20% and 70% to the overall programming grade. However, in all of the course outlines, the learning outcomes of the course were clearly linked to the objective of the assessment, thereby giving confidence that the assessments were designed to test the programming ability of the novice programmer.

There was little difference in the learning outcomes of each introductory programming course. The learning outcomes largely stated that the novice programmers would be able to perform a range of programming-related tasks such as creating, editing, compiling, running, debugging, and testing a program by the end of their course. The learning outcomes also stated that the novice programmer would be able to use a problem-solving strategy to design a solution to a programming problem. In some of the learning outcomes, the ability to use the standard programming library and using concepts that are appropriate to the problem scenario were also stated as a learning outcome for the introductory programming course. These learning outcomes provide strong evidence that the programming grade in the introductory programming courses that were sampled in this study had assessed programming-related activities.

9.7 Implications of this study

The findings from this study have implications for course instructors who design and deliver introductory programming courses. This thesis argues that course instructors should support the novice programmer's learning by making clear behavioural expectations and designing courses which stimulate and support effective behaviour. Specifically, course instructors should develop the self-efficacy beliefs of the novice programmers and encourage engagement behaviour that leads to better performance in their introductory programming course. This may be done by implementing the recommendations made in this thesis about the design and delivery of their introductory programming course. It should be noted that the introductory programming course outlines of the participating HEIs did not appear to show any indication of the behaviour that is expected of the novice programmers suggesting that there is a lack of emphasis on stimulating and supporting effective behaviour in introductory programming courses.

9.7.1 Implications of the hypotheses that were supported in this study

This study found that the self-efficacy beliefs of the novice programmers increased as they progressed in their introductory programming course particularly in the novice programmers with an average intelligence score and that *programming self-efficacy* influences all the engagement factors with the exception of help-seeking. In addition, this study found that e*ffort*, *enjoyment*, *deep learning*, and *surface learning* predict the novice programmer's *programming grade*.

Thus, based on the hypotheses that were supported in this study, course instructors should improve the design and delivery of introductory programming courses by designing programming tasks that build the self-efficacy beliefs of the novice programmer, by tactfully enabling the novice programmers to expend effort, by developing enjoyable programming tasks, by developing programming tasks that require deep learning, discourage surface learning, and finally by identifying poor to average performers early in the course so that remedial action may be taken.

Build self-efficacy

The first step in building the self-efficacy beliefs of the novice programmer is to understand the level of self-efficacy beliefs of the novice programmer by administering the programming self-efficacy scale that was used in this study. Course instructors should administer the programming self-efficacy scale at regular intervals during the introductory programming course, particularly after the novice programmers have completed an assessment or after difficult concepts are introduced in the course. The course instructor can then gauge the programming self-efficacy belief of the novice programmer and can take remedial action if their programming self-efficacy belief appears to be low.

Bandura (1977) argued that performance accomplishment (enactive mastery) is the most influential method of improving an individual's self-efficacy beliefs. As such, course instructors should design programming tasks that require the novice programmers to repeatedly master a specific programming concept. The repeated mastery of programming concepts increases the self-efficacy beliefs of the novice programmer.

In particular, course instructors should focus on designing programming tasks around threshold concepts, which is one factor that contributes to the difficulties of learning programming (Chapter 2, Section 2.1.4.1). Threshold concepts are troublesome concepts that may hinder learning as the novice programmers may become stuck, frustrated, lose interest in the subject, adopt a surface approach to learning, or even withdraw from the course (Boustedt et al., 2007; Sorva, 2010) which could result in low self-efficacy beliefs. Threshold concepts are typically difficult to master, but once the programming concept is understood, novice programmers will be able to progress to the next programming concept. By focusing on programming tasks that require novice programmers to repeatedly master threshold concepts, novice programmers are able to increase their self-efficacy beliefs in programming. Further, the novice programmers will be able to see immediate results upon completing their programming task, and experience an immediate feeling of gratification, which is an indicator of emotional engagement, unique to this study, and not found in other studies related to learning programming.

Tactfully enable novice programmers to expend effort

Secondly, this thesis recommends that course instructors tactfully enable novice programmers to expend effort during their introductory programming course by using formative programming assessments that require deliverables at regular intervals during the introductory programming course, and that build on one programming concept after the other. The recommendation to use formative programming assessments is made because the participants in the focus groups explained that they expended effort by staying on top of their programming assessments and made an effort to start and complete their programming assessments early. The recommendation to use formative programming assessments that build on one programming concept after the other is linked to the earlier discussion on building the self-efficacy beliefs of novice programmers by repeatedly mastering the threshold concepts in programming. Course instructors may also provide sufficient exercises for novice programmers to practise during their introductory programming course since practise was frequently mentioned in the focus groups as important when learning to program while effective time management and consistently working throughout the course were two other effort-related recommendations that contributed to at least those novice programmers who had obtained an excellent programming grade in their introductory programming course. Providing sufficient exercises for novice programmers to practise is also linked to the earlier recommendation to build the self-efficacy beliefs of the novice programmer through performance accomplishments.

Develop enjoyable programming tasks

Thirdly, the findings from the focus groups suggest that some novice programmers enjoy programming, and the findings from the survey suggest that the novice programmers who obtained an excellent programming grade enjoy programming. This implies that novice programmers who prefer courses that require a hands-on approach to learning would enjoy programming since it requires novice programmers to write code to solve programming problems. However, to encourage novice programmers who were not enjoying programming as much as their peers, particularly novice programmers who appear to be average or good performers, this thesis recommends that course instructors develop programming tasks or introduce learning interventions that are enjoyable. The learning interventions proposed in recently published literature on learning programming could be in the form of pair programming (Liebenberg et al., 2012; Maguire et al., 2014) or by introducing new programming tools (Bishop-Clark et al., 2007; Major et al., 2014).

Develop programming tasks that require deep learning

The fourth recommendation is to develop programming tasks that require deep learning. This study found that the strategies to learn programming differ from the strategies to learn in other courses. For example, the participants in the focus groups found that their introductory programming course required a hands-on approach to learning compared to other courses which required reading, memorisation, analysis and arguments (Chapter 5, Section 5.5). In addition, the participants in the focus groups explained that although they understood the programming concepts, the application of the concepts was not as straightforward as other courses.

Thus, this thesis makes four recommendations to develop programming tasks that require deep learning. The first recommendation is to develop programming tasks that require the novice programmers to reflect on how well they understand the programming problem and the logical flow of their proposed solution before coding the solution to the programming problem. The second recommendation is to conduct a programming assessment clinic for the purpose of reviewing the programming assessment criteria so that the novice programmers clearly understand the requirements of their programming assessment. The third recommendation is to design programming tasks that require novice programmers to think deeply about ways to resolve their programming problem, and the fourth recommendation is to devise a strategy to debug programming errors and is discussed in Section 9.7.2.

Discourage surface learning

The negative relationship between post-programming self-efficacy and surface learning and between surface learning and programming grade suggests that novice programmers were

clearly aware that surface learning would not lead to programming success. However, the discussion in Section 9.4.5 highlighted that those novice programmers who obtained an average programming grade had felt that their surface learning approach could lead to success in their introductory programming course. Thus, this thesis recommends that in addition to developing programming tasks that require deep learning, course instructors should discourage surface learning by explaining to novice programmers the differences between deep learning and surface learning so that the novice programmers who use a surface learning approach are aware that their learning approach would not lead to programming success.

Identify poor to average performers early

From the discussions in Section 9.4.2, 9.4.5, and 9.4.7, novice programmers with an average programming grade were clearly less engaged in their introductory programming course, resulting in the achievement of a poor to average programming grade. Thus, it is important that course instructors identify the poor to average performers early in the course since they are at a higher risk of becoming less engaged in the course, developing low self-efficacy beliefs, which may then lead to poor performance and eventually dropping out of the course. One way of identifying poor to average performers is linked to the earlier recommendation of tactfully enabling novice programmers to expend effort in their course by using formative programming assessments that require deliverables at regular intervals. In addition to enabling novice programmers to expend effort, the formative programming assessments enable course instructors to assess the novice programmer's performance early in the course and identify the programming concepts that novice programmers did not appear to fully understand. Appropriate remedial action in the form of additional mentoring sessions can then be organised in order to improve the performance of the novice programmer.

9.7.2 Implications of hypotheses that were not supported in this study

This study found that *programming self-efficacy* negatively influenced help-seeking. This finding was unexpected since existing literature on learning theory had found positive correlations between self-efficacy and help-seeking. In addition, this study found a negative and unexpected relationship between *persistence, help-seeking, trial and error, interest, gratification* and the *programming grade* of the novice programmer.

Like the hypotheses that were supported, the hypotheses that were not supported in this study have implications for course instructors who design and deliver introductory programming courses. This thesis recommends that course instructors make the novice programmers aware of the importance of using an instrumental help-seeking strategy, encourage novice programmers to persist strategically, and to devise a strategy to debug programming errors. In addition, this thesis argues that course instructors should advise novice programmers that affect-related interest and the feeling of gratification may not lead to better performance.

Make novice programmers aware of the importance of using an instrumental help-seeking strategy

From the discussion in Section 9.4.3, several plausible explanations were offered for the overall negative relationship between post-programming self-efficacy and help-seeking, and between help-seeking and programming grade. In particular, the plausible explanations that were offered suggest that novice programmers need to understand on how to seek help and the impact of their help-seeking strategy on their performance in the introductory programming course. Therefore, this thesis recommends that course instructors should make novice programmers aware of the importance of using an instrumental help-seeking strategy as opposed to using an executive help-seeking strategy. Course instructors should advise novice programmers that if they are unable to resolve their programming problem after receiving hints to the solution (instrumental help-seeking), they should take remedial action by reviewing the programming concepts that they did not understand or consult the course instructor for further mentoring. In addition, course instructors should advise novice programmers who help their peers to provide helpful hints and not solutions to the programming problem.

Encourage novice programmers to persist strategically

In Section 9.4.4, loss of productivity and the need to persist less were offered as plausible explanations for the negative relationship between persistence and programming grade. Although not statistically significant, the need to persist less appears to work favourably to novice programmers who obtained an excellent programming grade while the novice programmers who obtained a good or average programming grade and the novice programmers with an increase in their programming self-efficacy beliefs appear to perform better when they persisted in their introductory programming course. As such, this thesis recommends that course instructors should encourage novice programmers to persist, particularly those who appear to be facing difficulties when learning to program. Further, based on Bennedsen and Casperson's argument on the loss of productivity (Chapter 8, Section 8.3.1.2), course instructors should also encourage novice programmers to persist strategically by planning ahead in order to meet the deadline to complete the programming assessment and to persist only in programming related problems that have a large contribution to the programming assessment.

Devise a strategy to debug programming errors

From the discussion in Section 9.4.6, several plausible explanations were offered for the negative relationship between trial and error and programming grade. In particular, this thesis found that novice programmers with an average intelligence score had poor debugging skills and tended to use ineffective trial and error strategies to solve their programming errors. Further, evidence from the large body of research published in Computer Science-related

journals shows that the software industry places heavy emphasis on testing software applications and programmers typically use debugging tools and strategies to interpret and resolve their programming errors. This practice is clearly not emulated in introductory programming courses, leading to a trial and error strategy to debugging programming errors. As such, this thesis recommends that course instructors should devise a strategy that can help novice programmers debug their programming errors systematically. Course instructors should show the novice programmers how the debugging tool that is embedded in the programming environment (IDE) may be used to interpret their errors and devise a set of guidelines that explains how to solve the programming errors. In doing so, the novice programmers will have a strategy for debugging programming errors and will be able to avoid using ineffective trial and error strategies, leading to increased self-efficacy beliefs when learning programming.

Advise novice programmers that affect-related interest and gratification may not lead to better performance

Since interest and gratification did not predict programming grade, this thesis argues that course instructors should advise novice programmers that affect-related interest and the feeling of gratification may not lead to better performance. Course instructors must be aware that novice programmers who show interest in their introductory programming course may do so either due to value- or affect-related interest. As such, this thesis argues that course instructors should advise novice programmers of the difference between affect- and value-related interest and the impact of each type of interest on their performance.

On the other hand, since gratification is a new indicator of emotional engagement and the findings in this study suggests that further research is required to refine the construct, course instructors can, at this stage, advise novice programmers that their feeling of gratification may not necessarily lead to better performance. Instead, the novice programmers should review the assessment criteria of their programming assessment and ensure that they have met all the requirements to achieve a higher programming grade.

9.8 Chapter Summary

Further insights into the findings of this study were offered by: examining the extent in which the results may be generalised to the sample population, the impact of intelligence as a confounding variable, the predictability claims in this study, an in-depth examination of the indicators of engagement, the gender-based differences, and the validation of programming grade. The insights into the findings of this study then led to implications and recommendations to course instructors to improve the design and delivery of their introductory programming course.

Chapter 10: Conclusion

This thesis sets out a summary of the key findings and a summary of the implications of this study. The key findings are based on the results of the structural model that depicts the relationships between programming self-efficacy, the lower-order engagement factors (Chapter 8, Figure 8.1), and the programming performance of the novice programmer. This thesis discusses the theoretical and practical contributions of this study, the limitations of the study, and concludes by making recommendations for future research.

10.1 Summary of Research

This study set out to examine the relationship between programming self-efficacy, engagement and programming performance by developing and validating a model that explains the effect of the relationship between self-efficacy and the engagement factors, and the effect of the relationship between the engagement factors and the programming performance of novice programmers in introductory programming courses. The motivation for this study came from the on-going problem of high attrition and failure rates in introductory programming courses. Introductory programming courses provide important foundational concepts for novice programmers who wish to obtain a qualification in the field of Computer Science and other related fields, and who intend to pursue a career in software development. Despite more than two decades of research that have examined the cause for the high attrition and failure rates, and a plethora of solutions that have been proposed to improve the performance of novice programmers in introductory programming courses, there is still a lack of empirical evidence that suggests that the proposed solutions have been effective. Thus, the purpose of this study was to fill a gap in the published research on the teaching of computer programming by proposing and empirically testing self-efficacy and engagement, which are two behavioural factors that may predict the performance of novice programmers in introductory programming courses.

The existing literature on the teaching of computer programming argues that the demanding cognitive load in introductory programming courses is one factor that affects the performance of novice programmers. This is because novice programmers need to master multiple domains, use demanding programming languages, and are faced with a number of threshold concepts when learning programming (Chapter 2, Section 2.1.4.1). To overcome the cognitive load, solutions such as using educational technologies, re-structuring the pedagogical design, and using innovative artifacts to learn programming have been proposed but showed little evidence of its effectiveness (Chapter 2, Section 2.1.4.1).

The thesis then argues that the novice programmer's behaviour affects their performance in their introductory programming courses and appears to be an important predictor of programming performance although it has received little attention from researchers (Chapter

2, Section 2.1.4.2). Subsequently, by applying Hong et al.'s (2014) guidelines for developing context-specific theory, this thesis examined the literature on learning theory and computer programming and developed a research model that proposes a relationship between programming self-efficacy, engagement, and programming performance.

The engagement construct is a multi-dimensional construct which is made up of three higher-order constructs. They are *behavioural engagement*, *cognitive engagement*, and *emotional engagement*. Next, using the literature on learning theory and computer programming, and in consultation with introductory programming course instructors, the potential indicators of behavioural engagement, cognitive engagement and emotional engagement in introductory programming courses were identified. *Participation*, *effort*, *persistence*, and *help-seeking* were identified as potential indicators of behavioural engagement, *deep learning*, *surface learning*, and *trial and error* were identified as potential indicators of cognitive engagement while *interest* and *enjoyment* were identified as potential indicators of emotional engagement in introductory programming courses.

A three-phased mixed methods approach was employed to collect the data. The data was collected from novice programmers who were enrolled in introductory programming courses at Higher Educational Institutions (HEIs) in Malaysia and in New Zealand. The data collected from the HEIs in Malaysia and in New Zealand were able to offer an international perspective on the relationship between programming self-efficacy, engagement, and the programming performance of novice programmers in introductory programming courses. In addition, this study benefits from a fairly large heterogeneous group of participants (433 participants), enabling the findings to be generalised to the larger population of novice programmers.

A qualitative approach was used in the second phase of the data collection. Focus groups were held mid-way through the first introductory programming course to refine and validate the indicators of engagement in introductory programming courses. The participants in the focus groups provided rich narratives of the learning activities that they were engaged in during their introductory programming course, leading to strong evidence of the presence of the indicators of engagement that were initially proposed in the research model.

During the focus groups, g*ratification* emerged as a new indicator of emotional engagement in introductory programming courses. The findings from the focus groups suggested that novice programmers looked forward to receiving *immediate gratification* when they successfully debugged the errors in their programming assessment and were able to immediately see the outcome of their programming task. The findings of the focus groups also revealed that there were no significant differences in the engagement behaviour of novice programmers in the introductory programming courses in Malaysia and in New Zealand. On the other hand, during the scale development stage, *participation* was dropped from this study due to a lack of reliable measures.

A quantitative approach using online survey questionnaires was employed in the first and third phases of the data collection. During the first phase, data on the programming self-efficacy beliefs of the novice programmer was collected at the beginning of their introductory programming course (measured by pre-programming self-efficacy). During the third phase, data on the programming self-efficacy beliefs (measured by post-programming self-efficacy) of the novice programmer was collected again but at the end of their introductory programming course. In addition, data on the engagement behaviour of the novice programmer and their self-assessment of their programming performance was also collected in the third phase. The data on the programming grade of the novice programmers was collected between 6 and 8 weeks after the novice programmers completed their assessments in the course. Since the data was collected at three intervals and the self-efficacy of novice programmers had increased over time, and had strengthened the relationship between post-programming self-efficacy and the engagement of the novice programmers, the claims that post-programming self-efficacy influences the engagement of novice programmers, and that the engagement of the novice programmers predicts their programming performance could be established. The programming performance of the novice programmer was measured using the novice programmer's course grades and their self-assessment of their performance in the course.

This thesis also argued that in addition to programming self-efficacy and engagement, there may be other factors that may influence the programming performance of novice programmers. Therefore, *intelligence* and *prior programming experience* were identified as two confounding variables in this study. Intelligence was measured by the school result of the novice programmer. The findings in this thesis showed that intelligence and prior programming experience did not have a significant impact on the programming performance of the novice programmers. However, novice programmers with an average intelligence score showed a notably higher strength in the relationship between post-programming self-efficacy and the engagement of the novice programmers and are discussed in the Section 10.1.1. The analysis of the data from the survey questionnaires produced several key findings and is discussed in Section 10.1.1 and Section 10.1.2.

10.1.1 Research Question 1 (RQ1)

Table 10.1 summarises the hypotheses that answer Research Question 1 (RQ1). Pre-programming self-efficacy had a strong positive effect on post-programming self-efficacy (Hypothesis H1). This finding supports the findings from prior research that the self-efficacy beliefs of novice programmers increase as they progress in their introductory programming course. Interestingly, the increase in programming self-efficacy was significantly stronger in the novice programmers with an average intelligence score compared to the novice programmers with a good or excellent intelligence score, and is consistent with Bandura's (1977) Social Cognitive Theory which states that performance accomplishment is one source by which individuals may develop their self-efficacy beliefs (Chapter 2, Section 2.2.3). The

significantly higher increase in the programming self-efficacy beliefs of the novice programmers with an average intelligence score suggests that the novice programmers enter the introductory programming course with significantly lower self-efficacy beliefs due to their mediocre performance in school while the novice programmers with a good or excellent intelligence score had entered the course with strong self-efficacy beliefs since they performed very well in school and had subsequently expected to be successful in their introductory programming course, resulting in a lower increase in their self-efficacy beliefs compared to their peers who obtained an average intelligence score.

Table 10.1: Summary of hypotheses that answer research question 1 (RQ1)

Hypotheses that are supported		**Hypotheses that are NOT supported**	
Hypothesis	**Relationship**	**Hypothesis**	**Relationship**
H1	Pre-programming self-efficacy -> Post-programming self-efficacy	H3	Post-programming self-efficacy -> Help-seeking
H4	Post-programming self-efficacy -> Effort		
H5	Post-programming self-efficacy -> Persistence		
H6	Post-programming self-efficacy -> Deep Learning		
H7	Post-programming self-efficacy -> Surface Learning		
H8	Post-programming self-efficacy -> Trial and Error		
H9	Post-programming self-efficacy -> Interest		
H10	Post-programming self-efficacy -> Enjoyment		
H20	Post-programming self-efficacy -> Gratification		
Note: Hypothesis H2 was not tested since the participation construct was dropped from the research model during the instrument development phase due to lack of reliable measures.			

The findings that there was a strong positive relationship between programming self-efficacy and the engagement constructs (Hypotheses H4, H5, H6, H7, H8, H9, H10, and H20), with the exception of help-seeking (Hypothesis H3), answers Research Question 1 (RQ1). This finding strengthened the importance of self-efficacy on human behaviour and the importance of Bandura's self-efficacy theory within the context of novice programmers in introductory programming courses. On the other hand, the negative relationship between programming self-efficacy and help-seeking was unexpected. However, on further analysis, this study

found that novice programmers who obtained a good or excellent programming grade tended not to ask for help as their programming self-efficacy beliefs increased.

The strong positive relationship between programming self-efficacy and the engagement factors have implications for course instructors who are involved in the design and delivery of introductory programming course. This thesis argues that course instructors should gauge the self-efficacy beliefs of the novice programmers in introductory programming courses by administering the programming self-efficacy scale that was used in this study and take remedial action if their programming self-efficacy belief appears to be low. Course instructors should then design programming tasks that require the novice programmers to repeatedly master a specific programming concept. Repeated mastery of programming concepts increases the self-efficacy beliefs of the novice programmer and is consistent with Bandura's (1977) argument that performance accomplishment is the most influential method of improving an individual's self-efficacy beliefs.

10.1.2 Research Question 2 (RQ2)

Table 10.2 summarises the outcome of the hypotheses that were tested in this study in order to answer Research Question 2 (RQ2). A mixed set of outcomes were observed in the relationships between the novice programmer's engagement and their programming performance. As initially hypothesised, e*ffort*, *enjoyment*, and *deep learning* had positively predicted the novice programmer's programming performance, while *surface learning* negatively predicted the novice programmer's programming performance. Effort and enjoyment were ranked the highest in the Importance Performance-Matrix Analysis (IPMA), suggesting that effort and enjoyment are two engagement factors that are important predictors of success in introductory programming courses

The hypotheses that were supported in this study have implications for course instructors who are involved in the design and delivery of introductory programming courses. This thesis argues that course instructors should improve the design and delivery of introductory programming courses by focusing on developing strategies that will tactfully enable novice programmers to expend effort, by developing enjoyable programming tasks, by developing programming tasks that require deep learning, by discouraging surface learning, and finally by identifying poor to average performers early in the course so that remedial action may be taken.

With the exception of gratification, which is an indicator of emotional engagement, the effect of the indicators of engagement on the programming grade of the novice programmer was similar to the effect of the indicators of engagement on the novice programmer's assessment of their programming performance. This finding suggests that at the end of the introductory programming course, the novice programmers were aware of their level of engagement in the course, and had self-assessed their performance consistently with the actual grades that they had obtained in their introductory programming course.

Table 10.2: Summary of hypotheses that answer research question 2 (RQ2)

Hypotheses that are supported		**Hypotheses that are NOT supported**	
Hypothesis	**Relationship**	**Hypothesis**	**Relationship**
H13a	Effort -> Programming Grade	H12a	Help Seeking -> Programming Grade
H13b	Effort -> Self-Assessment	H12b	Help Seeking -> Self-Assessment
H15a	Deep Learning -> Programming Grade	H14a	Persistence -> Programming Grade
H15b	Deep Learning -> Self-Assessment	H14b	Persistence -> Self-Assessment
H16a	Surface Learning -> Programming Grade	H17a	Trial and Error -> Programming Grade
H16b	Surface Learning -> Self-Assessment	H17b	Trial and Error -> Self-Assessment
H19a	Enjoyment -> Programming Grade	H18a	Interest -> Programming Grade
H19b	Enjoyment -> Self-Assessment	H18b	Interest -> Self-Assessment
H21b	Gratification -> Self-Assessment	H21a	Gratification -> Programming Grade
Note: Hypotheses H14a and H14b were not tested since the participation construct was dropped from the research model during the instrument development phase due to lack of reliable measures.			

There was a negative relationship between *help-seeking* and programming performance. The negative relationship contradicted the initial hypothesis of a positive relationship between help-seeking and programming performance. However, in retrospect, evidence from recent literature on computer programming suggests that the higher level of acceptability to ask for help in programming courses compared to other courses (Aasheim et al., 2012), and the lack of context specific help in programming (Li et al., 2013) (Chapter 8, Section 8.3.1) may have led to the negative relationship. On further analysis, the lack of clarity in the wording of the items that measure help-seeking may have also contributed to the negative finding. The items were designed to measure instrumental help-seeking strategies, whereby students who use an instrumental help-seeking strategy expect hints and not an answer to solve the problem. Expecting an answer to solve a problem is reflective of an executive help-seeking strategy. However, the wording of the items may not have made a clear distinction between executive help-seeking strategy and instrumental help-seeking strategy.

Similar to the findings on help-seeking, there was a negative relationship between *persistence* and programming performance. Persistence is an indicator of behavioural engagement. Here again, in retrospect, evidence from the recent literature on computer programming provided a plausible explanation for the negative relationship. The novice programmers may have experienced a loss of productivity when they persisted in resolving the errors in their programming assessments, which then resulted in a negative effect on their programming grade. On further examination of the group-specific results, the negative relationship was specific to novice programmers with an excellent programming grade and novice programmers who had an increase in their self-efficacy beliefs during their introductory programming course. The negative finding suggests that novice programmers who excel in programming might find their assessments easier and require less persistence as they may have a solid understanding of programming concepts may be able to program with fewer errors, and may be able to debug their programming errors without much difficulty.

There was also a negative relationship between *interest* and programming performance. Upon further examination of the items that measure interest, this study found that interest was measured as an affect-related interest. Evidence from the recent literature on learning theory suggests that affect-related interest may not lead to programming success compared to value-related interest (O'Keefe & Linnenbrink-Garcia, 2014).

On the other hand, *trial and error* and *gratification* are two new indicators of engagement that had emerged from this study. Trial and error and gratification appear to be unique to the context of introductory programming courses. However, the effect of trial and error and gratification on the programming performance of the novice programmer was not significant, with the exception of the relationship between gratification and self-assessment.

The measures for trial and error and poor debugging skills are two plausible explanations for the not significant relationship between trial and error and the programming performance of the novice programmer. The items to measure trial and error did not clearly distinguish between effective and ineffective trial and error strategies when solving programming problems, and in particular, when debugging programming errors, which may have led to the not significant relationship. The third plausible explanation for the weak relationship between trial and error and programming performance could be due to the poor debugging skills, particularly in the novice programmers with an average intelligence score. A literature search on debugging tools confirmed that novice programmers found debugging a challenge and that several debugging tools have been designed to overcome the difficulty of debugging programming errors. Additionally, the course outlines from the introductory programming courses that were sampled in this study showed little to no emphasis on interpreting programming errors and the strategies that may be used to debug programming errors. However, evidence from the large body of research published in Computer Science-related journals showed that the software industry placed heavy emphasis on testing software applications and programmers typically used debugging tools and strategies to interpret and

resolve their programming errors. This practice is clearly not emulated in the introductory programming courses that were sampled in this study, and which could have resulted in the novice programmers using a trial and error strategy to debug their programming errors.

The positive relationship between gratification and self-assessment but negative and not significant relationship between gratification and programming grade shows a disconnect between the novice programmer's judgment of their performance and the achievement of a better programming grade when they felt gratified upon seeing the output of their program. One plausible explanation for the disconnect could be that although the novice programmers were able to see the output of their program and make their program work without errors, they may not have met all the requirements of the programming assessment that would lead to a better programming grade.

Like the hypotheses that were supported in this study, it is possible to argue that the hypotheses that were not supported in this study also have implications for course instructors who are involved in the design and delivery of introductory programming courses. This thesis argues that course instructors should improve the design and delivery of introductory programming courses by using strategies to educate novice programmers on the importance of using an instrumental help-seeking strategy, encourage novice programmers to persist strategically, and to devise a strategy to debug programming errors. In addition, this thesis argues that course instructors should advise novice programmers that affect-related interest and the feeling of gratification may not lead to better performance.

10.2 Contributions of the Research

In theory, this study contributes to the existing body of knowledge on the teaching of computer programming and to the literature on learning theory, and in practice, this study contributes to the design and delivery of introductory programming courses.

10.2.1 Contribution to Theory

Validation and operationalization of research model. The first contribution to theory this thesis makes is the validation and operationalisation of a research model (Chapter 5, Section 5.4) that explains the effect of programming self-efficacy on the novice programmer's engagement, and the effect of the novice programmer's engagement on their programming performance. By applying the guidelines for developing context-specific theory in IS research that was proposed by Hong et al. (2014), this thesis examined prior theory on self-efficacy and student engagement and developed a research model with testable hypotheses for examining the relationship between programming self-efficacy, a set of engagement factors, and the programming performance of novice programmers in introductory programming courses. The research model was then refined by identifying and confirming the engagement factors that were specific to novice programmers in introductory programming courses by reviewing prior empirical research on introductory programming courses and by conducting focus groups. The hypotheses were then tested using a survey

questionnaire on a representative sample of novice programmers who were enrolled in introductory programming courses in Malaysia and in New Zealand.

From the analysis, a final research model that explains the influence of the novice programmer's programming self-efficacy beliefs on their engagement, and that explains the predictability of the relationship between the novice programmer's engagement and their programming performance was developed (Chapter 8, Section 8.5). This contribution to theory is classified as Type IV theory in IS research (Gregor, 2006). The Type IV theory "provides prediction, has testable propositions, and causal explanations" (Gregor, 2006, p. 620). The development of Type IV theories is also strongly supported by Weber (2012).

New context-specific engagement constructs. Hong et al. (2014) proposed that identifying context-specific factors is one way of developing context-specific theory in IS research. Therefore, the second contribution to theory that this thesis makes is in the identification of *trial and error* and *gratification* as two new indicators of engagement that appear to be specific to the context of novice programmers in introductory programming courses. During the development of the proposed research model (Chapter 3), the existing literature on learning theory and computer programming was reviewed for possible engagement factors in introductory programming courses. Trial and error was identified as a potential indicator of cognitive engagement in introductory programming courses. When the hypotheses were tested, there was a strong positive relationship between programming self-efficacy and trial and error. However, the relationship between trial and error and programming performance was not statistically significant.

A similar finding was observed with gratification, which is an indicator of emotional engagement. This indicator emerged from the focus groups and there was a strong positive relationship between programming self-efficacy and gratification, and between gratification and self-assessment. However, the relationship between gratification and programming grade was not statistically significant. As new indicators (trial and error and gratification) of engagement, the not statistically significant findings should not be a cause to exclude these indicators from the theory of student engagement and computer programming. Instead, recommendations are made to continue examining these two constructs within the context of introductory programming courses (Section 10.4) due to the strong evidence of the presence of trial and error and gratification in the literature on computer programming, and from the focus groups.

10.2.2 Contribution to Practice

Explanation of the relationships. The first contribution to practice is the explanation of the relationships in this study. The strong positive relationship between pre- and post-programming self-efficacy confirmed that the self-efficacy beliefs of the novice programmers increased with instruction. The programming self-efficacy beliefs of the novice programmers then positively influenced their engagement in the course, with the exception of the

relationship between programming self-efficacy and help-seeking. Next, the findings of the focus groups confirmed the indicators of engagement in introductory programming courses and are discussed in Chapter 5.

The novice programmer's engagement in the introductory programming course then predicted their programming performance. However, not all the engagement factors had predicted the programming performance of the novice programmer. Help-seeking, persistence, interest, trial and error, and gratification (applies to programming grade only) did not predict the programming performance of the novice programmer and plausible reasons for these were established in Chapter 8 and Chapter 9.

On the other hand, effort, enjoyment, deep learning and surface learning were found to predict the programming performance of the novice programmer. The findings also showed that using a surface learning approach caused novice programmers to perform poorly in their introductory programming course.

Resulting survey instrument. The second contribution of this study is the resulting survey instrument. The survey questionnaire in Phase 3 of this study was developed using a comprehensive three-stage instrument development process that was proposed by Moore and Benbasat (1991) and was subjected to rigorous reliability and validity tests. At present, there appears to be no evidence of a comprehensive survey instrument that measures the novice programmer's engagement in introductory programming courses. However, there are survey instruments for assessing general student engagement such as the NSSE (Kuh, 2009; "About NSSE," n.d.) and the AUSSE (Coates, 2010) and are widely used in the USA, Canada, and Australia. Several other student engagement survey instruments have also been used and are discussed in Chapter 2, Section 2.3.4. The wide use of these survey instruments provides strong evidence of the importance of understanding student engagement. Similarly, the resulting survey instrument from this study may be used by course instructors and faculty members to examine the engagement behaviour of novice programmers in introductory programming courses, which leads to the third contribution to practice.

Design and deliver introductory programming courses that make clear behavioural expectations and which stimulate and support effective behaviour. The rich narratives provided by the focus group participants revealed and confirmed several engagement factors that are specific to introductory programming courses. The findings from the survey questionnaire then provided empirical evidence of the relationship between programming self-efficacy, engagement, and the programming performance of the novice programmer. Thus, this thesis argues that the findings have implications for course instructors who are involved in the design and delivery of introductory programming courses. This thesis then makes recommendations for course instructors to design and deliver courses that are directed at increasing the programming self-efficacy beliefs of the novice programmer and to

encourage successful engagement behaviour. The implication and recommendations are discussed in Chapter 9, Section 9.7.

10.3 Limitations of the Research

Although several limitations were encountered at each stage of the research, most were resolved. But, the following limitations had to be accepted:

Fewer focus groups in Malaysia. Due to the limited time to collect data in Malaysia which coincided with the end of the year term break, and religious holidays at the start of the following year, only one focus group was conducted in Malaysia. However, the findings did not reveal any significant differences between the focus groups in New Zealand and the focus group in Malaysia.

Fewer HEIs participated in Malaysia. Due to the enforcement of the PDPA (Personal Data Protection Act) at the time of the data collection in Malaysia, several HEIs that were approached had declined to participate in the research due to concerns about the confidentiality of their student data. Thus, HEIs in Malaysia had to be contacted through professional connections resulting in only 9 HEIs contacted in Malaysia compared to 19 HEIs that were contacted in New Zealand.

Design and delivery of courses not examined. During data collection, the design and delivery of the introductory programming courses were not examined since the purpose of the study was to examine the behaviour of the novice programmers without influencing the pedagogy used in of the introductory programming courses. As such, it may be possible that factors such as teaching methods and the complexity of the programming assessments may have influenced the programming performance of the novice programmer, and the recommendations made to course instructors in this thesis may already be in practice in some of the HEIs that were sampled in this study.

10.4 Recommendations for Future Research

Recommendations for future research are made based on the opportunities that emerged from the findings of this study. Firstly, the role of participation in an introductory programming course requires further research. The participation construct was dropped during the scale development stage due to a lack of reliable measures. Since there appears to be a dearth of research that examines the links between programming self-efficacy, participation, and the programming performance of the novice programmer, researchers may consider conducting interviews or focus groups that will allow the identification of a set of activities that assesses participation. The participation construct should also be clearly conceptualised by establishing the method of instruction, institutional policies, and assessment methods so that the items that are developed to measure participation in the course are reliable.

Secondly, help-seeking and persistence were found to have a negative effect on the programming performance of the novice programmer. By taking into account the plausible explanations offered for the negative relationship, help-seeking and persistence could be re-conceptualised and tested on a new sample of novice programmers in introductory programming courses.

Thirdly, this study was undertaken using the behavioural science paradigm. Future research could focus on examining the relationships between self-efficacy, engagement and programming performance using the design science paradigm. Researchers using the design science paradigm have developed software artifacts that are able to observe the behaviour of the user without being intrusive. These software artifacts could be used to examine the engagement behaviour of novice programmers when they are writing codes for their programming assessment.

Next, the two new constructs, trial and error and gratification appear to be promising indicators of cognitive engagement and emotional engagement respectively. The findings show that these two indicators of engagement support Research Question 1 but not Research Question 2. The findings show strong positive relationships between programming self-efficacy and the engagement constructs. However, the effect of these engagement constructs on the programming grade of the novice programmer was not statistically significant. Thus, researchers could conduct further research to examine the importance of these two indicators of engagement within the context of introductory programming courses.

Lastly, experiments could be carried out to determine the effect of implementing the recommendations made to course instructors to improve the design and delivery of introductory programming courses.

In conclusion, it is hoped that the recommendations that emerged from the findings of this study and the recommendations made for future research would contribute to the understanding of the behaviour of novice programmers in introductory programming courses. In addition, it is hoped that the recommendations would provide guidance to course instructors to make clear behavioural expectations and design courses which stimulate and support effective behaviour. In doing so, the novice programmers may be able to improve their performance and the failure and attrition rates in introductory programming courses may be reduced.

References

Aasheim, C. L., Rutner, P. S., Li, L., & Williams, S. R. (2012). Plagiarism and Programming: A Survey of Student Attitudes. *Journal of Information Systems Education*, *23*(3), 297-313.

About NSSE (n.d.). NSSE: National Survey of Student Engagement. Retrieved 27 February 2015, from http://nsse.iub.edu/

ACM/IEEE-CS Joint Interim Review Task Force. Computer Science Curriculum 2008: An Interim Revision of CS 2001. (2008). *Report from the Interim Review Task Force*. Retrieved from http://www.acm.org/education/curricula/ComputerScience2008.pdf

Agarwal, R., Sambamurthy, V., & Stair, R. M. (2000). Research Report: The Evolving Relationship Between General and Specific Computer Self-Efficacy--An Empirical Assessment. *Information Systems Research, 11*(4), 418.

Akbulut, A. Y., & Looney, C. A. (2007). Inspiring students to pursue computing degrees. *Communications of the ACM*, *50*(10), 67-71. doi: http://dx.doi.org/10.1145/1290958.1290964

Alexander, S., Clark, M., Loose, K., Amillo, J., Daniels, M., Boyle, R., . . . Shinners-Kennedy, D. (2003). Case studies in admissions to and early performance in computer science degrees. *SIGCSE Bulletin*, *35*(4), 137-147. doi: 10.1145/960492.960541

Alice. (n.d.). Retrieved 2 February 2013, from http://alice.org

Alvarado, C., Lee, C. B., & Gillespie, G. (2014). *New CS1 pedagogies and curriculum, the same success factors?* Paper presented at the Proceedings of the 45th ACM technical symposium on Computer science education, Atlanta, Georgia, USA.

Ames, R., & Lau, S. (1982). An attributional analysis of student help-seeking in academic settings. *Journal of Educational Psychology, 74*(3), 414-423. doi: http://dx.doi.org/10.1037/0022-0663.74.3.414

Appleton, J. J., Christenson, S. L., Kim, D., & Reschly, A. L. (2006). Measuring cognitive and psychological engagement: Validation of the Student Engagement Instrument. *Journal of School Psychology*, *44*(5), 427-445. doi: http://dx.doi.org/10.1016/j.jsp.2006.04.002

Appleton, J. J., Christenson, S. L., & Furlong, M. J. (2008). Student engagement with school: Critical conceptual and methodological issues of the construct. *Psychology in the Schools*, *45*(5), 369-386. doi: 10.1002/pits.20303

Askar, P., & Davenport, D. (2009). An Investigation of factors related to self-efficacy for Java programming among Engineering students. *The Turkish Online Journal of Educational Technology – TOJET, 8*(1).

Atkinson, J. W. (1957). Motivational determinants of risk-taking behaviour. *Psychological Review, 64*, 359-372.

Bagozzi, R. P., & Phillips, L. W. (1982). Representing and Testing Organizational Theories: A Holistic Construal. *Administrative Science Quarterly*, *27*(3), 459-489.

Bagozzi, R. P., Yi, Y., & Phillips, L. W. (1991). Assessing Construct Validity in Organizational Research. *Administrative Science Quarterly*, *36*(3), 421-458. doi: 10.2307/2393203

Bandura, A. (1977). Self-efficacy: Toward a unifying theory of behavioral change. *Psychological Review, 84*(2), 191-215. doi: 10.1037/0033-295x.84.2.191

Bandura, A. (1977a). *Social Learning Theory*. Englewood Cliffs, New Jersey: Prentice Hall.

Bandura, A., Adams, N. E., & Beyer, J. (1977). Cognitive Processes Mediating Behavioral Change. *Journal of Personality and Social Psychology, 35*(3), pp. 125- 139.

Bandura, A., Adams, N. E., Hardy, A. B., & Howells, G. N. (1980). Tests of the generality of self-efficacy theory. *Cognitive Therapy and Research, 4*(1), 39-66.

Bandura, A. (1986). *Social Foundations of Thought and Action.* Englewood Cliffs, NJ: Prentice Hall.

Bandura, A. (1997). *Self-efficacy: The exercise of control.* New York: W H Freeman and Company.

Bandura, A. (2012). On the Functional Properties of Perceived Self-Efficacy Revisited. *Journal of Management, 38*(1), 9-44. doi: 10.1177/0149206311410606

Barlett, J. E., Kotrlik, J. W., & Higgins, C. C. (2001). Organizational research: Determining appropriate sample size in survey research. *Information Technology, Learning, and Performance Journal, 19*(1), 43-50.

Baron, R. M., & Kenny, D. A. (1986). The moderator–mediator variable distinction in social psychological research: Conceptual, strategic, and statistical considerations. *Journal of Personality and Social Psychology, 51*(6), 1173-1182. doi: 10.1037/0022-3514.51.6.1173

Bednarik, R., & Tukiainen, M. (2004). *Visual attention tracking during program debugging.* Paper presented at the Proceedings of the third Nordic conference on Human-computer interaction, Tampere, Finland.

Bembenutty, H. (2011). Academic delay of gratification and academic achievement. *New Directions for Teaching and Learning,* (126), 55-65. doi: 10.1002/tl.444

Bembenutty, H., & White, M. C. (2013). Academic performance and satisfaction with homework completion among college students. *Learning & Individual Differences, 24*, 83-88. doi: 10.1016/j.lindif.2012.10.013

Bennedsen, J., & Caspersen, M. E. (2006). Abstraction ability as an indicator of success for learning object-oriented programming? *SIGCSE Bulletin, 38*(2), 39-43. doi: 10.1145/1138403.1138430

Bennedsen, J., & Caspersen, M. E. (2007). Failure rates in introductory programming. *SIGCSE Bulletin, 39*(2), 32-36. doi: 10.1145/1272848.1272879

Bennedsen, J., & Caspersen, M. E. (2012). Persistence of elementary programming skills. *Computer Science Education, 22*(2), 81-107. doi: 10.1080/08993408.2012.692911

Bergin, S., & Reilly, R. (2006). Predicting introductory programming performance: A multi-institutional multivariate study. *Computer Science Education, 16*(4), 303-323. doi: 10.1080/08993400600997096

Betts, J. E., Appleton, J. J., Reschly, A. L., Christenson, S. L., & Huebner, E. S. (2010). A study of the factorial invariance of the Student Engagement Instrument (SEI): Results from middle and high school students. *School Psychology Quarterly, 25*(2), 84-93.

Biggs, J. B. (1987). *Student Approaches to Learning and Studying.* Hawthorn, Victoria: Australian Council for Educational Research.

Bishop-Clark, C., Courte, J., Evans, D., & Howard, E. V. (2007). A Quantitative and Qualitative Investigation of Using Alice Programming to Improve Confidence, Enjoyment and Achievement Among Non-Majors. *Journal of Educational Computing Research, 37*(2), 193-207.

Blackwell, A. F. (1996). *Metacognitive theories of visual programming: what do we think we are doing?* Paper presented at the IEEE Symposium on Visual Languages, 1996.

Blackwell, A. F. (2002). What is programming? In J. Kuljis, L. Baldwin & R. Scoble (Eds.), *In Proceedings of the 14th Workshop of the Psychology of Programming Interest Group (PPIG)* (pp. 204-218). Brunel University.

Blikstein, P. (2011). *Using learning analytics to assess students' behavior in open-ended programming tasks.* Paper presented at the Proceedings of the 1st International Conference on Learning Analytics and Knowledge, Banff, Alberta, Canada.

Boustedt, J., Eckerdal, A., McCartney, R., Mostrom, J. E., Ratcliffe, M., Sanders, K., & Zander, C. (2007). *Threshold concepts in computer science: do they exist and are they useful?* Paper presented at the Proceedings of the 38th SIGCSE technical symposium on Computer science education, Covington, Kentucky, USA.

Bresó, E., Schaufeli, W., & Salanova, M. (2011). Can a self-efficacy-based intervention decrease burnout, increase engagement, and enhance performance? A quasi-experimental study. *Higher Education, 61*(4), 339-355. doi: 10.1007/s10734-010-9334-6

Brown, S. D., Tramayne, S., Hoxha, D., Telander, K., Fan, X., & Lent, R. W. (2008). Social cognitive predictors of college students' academic performance and persistence: A meta-analytic path analysis. *Journal of Vocational Behavior, 72*(3), 298-308. doi: http://dx.doi.org/10.1016/j.jvb.2007.09.003

Bryman, A. (2008). *Social Research Methods* (3rd ed.). New York: Oxford University Press Inc.

Bureau of Labor Statistics, U. S. Department of Labour. (2014). Software Developers. *Occupational Outlook Handbook, 2014-15 Edition*. Retrieved 16 May 2015, from http://www.bls.gov/ooh/computer-and-information-technology/software-developers.htm

Burt, R. S. (1976). Interpretational Confounding of Unobserved Variables in Structural Equation Models. *Sociological Methods & Research, 5*(1), 3-52. doi: 10.1177/004912417600500101

Bye, D., Pushkar, D., & Conway, M. (2007). Motivation, Interest, and Positive Affect in Traditional and Nontraditional Undergraduate Students. *Adult Education Quarterly, 57*(2), 141-158.

Campbell, C., & Cabrera, A. (2014). Making the Mark: Are Grades and Deep Learning Related? *Research in Higher Education, 55*(5), 494-507. doi: 10.1007/s11162-013-9323-6

Carbone, A., Hurst, J., Mitchell, I., & Gunstone, D. (2009). *An exploration of internal factors influencing student learning of programming.* Paper presented at the Proceedings of the Eleventh Australasian Conference on Computing Education - Volume 95, Wellington, New Zealand.

Carini, R. M., Kuh, G. D., & Klein, S. P. (2006). Student Engagement and Student Learning: Testing the Linkages. *Research in Higher Education, 47*(1), 1-32.

Carlson, J. L., & Skaggs, N. T. (2000). Learning by Trial and Error: A Case for Moot Courts. *Journal of Economic Education*, *31*(2), 145-155.

Caspersen, M. E., Larsen, K. D., & Bennedsen, J. (2007). Mental models and programming aptitude. *SIGCSE Bulletin*, *39*(3), 206-210. doi: 10.1145/1269900.1268845

Chamillard, A. T. (2006). Introductory game creation: no programming required. *SIGCSE Bulletin*, *38*(1), 515-519. doi: 10.1145/1124706.1121502

Chen, P., & McGrath, D. (2003). Moments of Joy: Student Engagement and Conceptual Learning in the Design of Hypermedia Documents. *Journal of Research on Technology in Education*, *35*(3), 402.

Chilton, M. A., & Riemenschneider, C. K. (2000). *Investigating Computer Self-Efficacy with Students in COBOL Programming.* Paper presented at the AMCIS 2000 Proceedings. Paper 232.

Chin, W. W. (1998). Commentary: Issues and Opinion on Structural Equation Modeling. *MIS Quarterly*, *22*(1), vii-xvi. doi: 10.2307/249674

Chin, W. W. (1998a). The partial least squares approach to structural equation modeling. In G. A. Marcoulides (Ed.), *Modern Methods for Business Research* (pp. 295–336). Mahwah, NJ: Lawrence Erlbaum Associates.

Chin, W. W. (2003). *PLS Graph 3.0.* Houston: Soft Modeling Inc.

Christenson, S. L., Reschly, A. L., Appleton, J. J., Berman, S., Spanjers, D., & Varro, P. (2008). Best practices in fostering student engagement. In A. Thomas & J. Grimes (Eds.), *Best Practices in School Psychology* (5th ed., pp. 1099-1119). Bethesda, MD: National Association of School Psychologists.

Christenson, S. L., Reschly, A. L., & Wylie, C. (2012). *Handbook of Research on Student Engagement.* Retrieved from http://VUW.eblib.com/patron/FullRecord.aspx?p=884351.

Coates, H. (2010). Development of the Australasian Survey of Student Engagement (AUSSE). Higher Education: *The International Journal of Higher Education and Educational Planning*, *60*(1), 1-17.

Cohen, J. A. (1960). A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement*, *20*, 37-46.

Cohen, J. (1988). *Statistical Power Analysis For The Behavioral Science.* Mahwah, NJ: Lawrence Erlbaum.

Cohen, J. (1992). A Power Primer. *Psychological Bulletin*, *112*(1), 155-159.

Compeau, D. R., & Higgins, C. A. (1995). Computer Self-Efficacy: Development of a Measure and Initial Test. *MIS Quarterly*, *19*(2), 189-211.

Compeau, D., Higgins, C. A., & Huff, S. (1999). Social Cognitive Theory and Individual Reactions to Computing Technology: A Longitudinal Study. *MIS Quarterly*, *23*(2), 145-158.

Comrey, A. L., & Lee, H. B. (1992). *A First Course in Factor Analysis* (2nd ed.). Hillsdale, NJ: Lawrence Erlbaum Associates.

Corney, M., Teague, D., & Thomas, R. N. (2010). *Engaging students in programming.* Paper presented at the Proceedings of the 12th Australasian Conference on Computing Education - Volume 103, Brisbane, Australia.

Cronbach, L. J. (1971). Test Validation. In R. L. Thorndike (Ed.), *Educational Measurement* (2nd ed., pp. 443-507). Washington, DC: American Council on Education.

Crotty, M. (1998). *The foundations of social research: meaning and perspective in the research process.* Crows Nest, NSW: Allen & Unwin.

Dahotre, A., Krishnamoorthy, V., Corley, M., & Scaffidi, C. (2011). Using intelligent tutors to enhance student learning of application programming interfaces. *Journal of Computing Sciences in Colleges, 27*(1), 195-201.

Davies, P. (2006). Threshold Concepts: How can we recognise them? In J. H. F. Meyer & R. Land (Eds.), *Overcoming Barriers to Student Understanding: Threshold Concepts and Troublesome Knowledge.* New York: Routledge.

Davis, W. A. (1982). A Causal Theory of Enjoyment. *Mind, 91*(362), 240-256. doi: 10.2307/2253480

Davison, A. C., & Hinkley, D. V. (1997). *Bootstrap methods and their application.* Cambridge, UK: Cambridge University Press.

deBry, R. (2011). A learning space for beginning programming students. *Journal of Computing Sciences in Colleges, 27*(2), 4-11.

Deci, E. L., & Ryan, R. M. (2000). The "What" and "Why" of Goal Pursuits: Human Needs and the Self-Determination of Behavior. *Psychological Inquiry, 11*(4), 227-268. doi: 10.1207/s15327965pli1104_01

Deek, F. P., Kimmel, H., & McHugh, J. A. (1998). Pedagogical changes in the delivery of the first-course in computer science: Problem solving, then programming. *Journal of Engineering Education, 87*(3), 313-313.

Denzin, N. K., & Lincoln, Y. S. (Eds.). (2005). *The SAGE Handbook of Qualitative Research* (3rd ed.). California: Sage.

Denzin, N. K., & Lincoln, Y. S. (Eds.). (2011). *The SAGE Handbook of Qualitative Research* (4th ed.). California: Sage.

de Raadt, M., Hamilton, M., Lister, R., Tutty, J., Baker, B., Box, I., . . . Tolhurst, D. (2005). *Approaches to learning in computer programming students and their effect on success.* Paper presented at the 28th HERDSA Annual Conference: Higher Education in a Changing World (HERDSA 2005), Sydney, Australia. Retrieved from http://eprints.usq.edu.au/780/

DeVellis, R. F. (2012). *Scale Development: Theory and Applications* (3rd ed.). Thousand Oaks, California: SAGE Publications.

Deviney, D. E., Crawford, J., & Elder, K. L. (2012). Classroom antics: Fun with a purpose. *Journal of Instructional Pedagogies, 10*, 105-110.

Diamantopoulos, A. (2006). The error term in formative measurement models: interpretation and modeling implications. *Journal of Modelling in Management, 1*(1), 7-17. doi: doi:10.1108/17465660610667775

Diamantopoulos, A., Sarstedt, M., Fuchs, C., Wilczynski, P., & Kaiser, S. (2012). Guidelines for choosing between multi-item and single-item scales for construct measurement: a predictive validity perspective. *Journal of the Academy of Marketing Science, 40*(3), 434-449. doi: 10.1007/s11747-011-0300-3

Dillon, E., Anderson, M., & Brown, M. (2012). Comparing feature assistance between programming environments and their "effect" on novice programmers. *Journal of Computing Sciences in Colleges, 27*(5), 69-77.

Diseth, Å., Pallesen, S., Brunborg, G., & Larsen, S. (2010). Academic achievement among first semester undergraduate psychology students: the role of course experience, effort, motives and learning strategies. *Higher Education*, *59*(3), 335-352. doi: 10.1007/s10734-009-9251-8

Dorn, B., & Guzdial, M. (2010). *Learning on the job: characterizing the programming knowledge and learning strategies of web designers.* Paper presented at the Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Atlanta, Georgia, USA.

Doubé, W., & Lang, C. (2012). Gender and stereotypes in motivation to study computer programming for careers in multimedia. *Computer Science Education*, *22*(1), 63-78. doi: 10.1080/08993408.2012.666038

Dougherty, L. M., Abe, J., & Izard, C. E. (1996). Differential emotions theory and emotional development in adulthood and later life. In C. Magai & S. H. McFadden (Eds.), *Handbook of emotion, adult development, and aging* (pp. 27-41). San Diego, CA: Academic Press.

duBoulay, B. (1986). Some difficulties of learning to program. *Journal of Educational Computing Research*, *2*(1), 57-73.

Dupeyrat, C., & Mariné, C. (2005). Implicit theories of intelligence, goal orientation, cognitive engagement, and achievement: A test of Dweck's model with returning to school adults. *Contemporary Educational Psychology*, *30*(1), 43-59. doi: http://dx.doi.org/10.1016/j.cedpsych.2004.01.007

Ebrahimi, A. (2012). How Does Early Feedback in an Online Programming Course Change Problem Solving? *Journal of Educational Technology Systems*, *40*(4), 371-379.

Eccles, J. (1983). Expectancies, values, and academic behaviours. In J. T. Spence (Ed.), *Achievement and achievement motives: Psychological and sociological approaches* (pp. pp 75-146). San Francisco: Freeman.

Eckerdal, A., & Berglund, A. (2005). *What does it take to learn 'programming thinking'?* Paper presented at the Proceedings of the first international workshop on Computing education research, Seattle, WA, USA.

Eckerdal, A., McCartney, R., Mostrom, J. E., Ratcliffe, M., Sanders, K., & Zander, C. (2006). Putting threshold concepts into context in computer science education. *SIGCSE Bulletin*, *38*(3), 103-107. doi: 10.1145/1140123.1140154

Edwards, S. H. (2004). Using software testing to move students from trial-and-error to reflection-in-action. *SIGCSE Bulletin*, *36*(1), 26-30. doi: 10.1145/1028174.971312

Ehrenberg, M. F., Cox, D. N., & Koopman, R. F. (1991). The relationship between self-efficacy and depression in adolescents. *Adolescence*, *26*(102), 361-374.

Falchikov, N., & Boud, D. (1989). Student Self-Assessment in Higher Education: A Meta-Analysis. *Review of Educational Research*, *59*(4), 395-395.

Felder, R. M., & Silverman, L. K. (1988). Learning and Teaching Styles in Engineering Education. *Engineering Education & Training*, *78*(7), 674-681.

Fenollar, P., Román, S., & Cuestas, P. J. (2007). University students' academic performance: An integrative conceptual framework and empirical analysis. *British Journal of Educational Psychology*, *77*(4), 873-891. doi: 10.1348/000709907x189118

Field, A. (2013). *Discovering Statistics Using IBP SPSS Statistics* (4th ed.). Thousand Oaks, California: SAGE Publications Inc.

Finn, J. D. (1989). Withdrawing From School. *Review of Educational Research*, *59*(2), 117-142. doi: 10.3102/00346543059002117

Finn, J. D., & Zimmer, K. S. (2012). Student Engagement: What Is It? Why Does It Matter? In S. L. Christenson, A. L. Reschly & C. Wylie (Eds.), Handbook of Research on Student Engagement. Dordrecht: Springer Science+Business Media New York. Retrieved from http://VUW.eblib.com/patron/FullRecord.aspx?p=884351.

Flynn, J. L. (1984). The Mean IQ of Americans: Massive Gains 1932 to 1978. *Psychological Bulletin*, *95*(1), 29-51.

Ford, M., & Venema, S. (2010). Assessing the Success of an Introductory Programming Course. *Journal of Information Technology Education*, *9*, 133-145.

Fornell, C., & Larcker, D. F. (1981). Evaluating Structural Equation Models with Unobservable Variables and Measurement Error. *Journal of Marketing Research*, *18*(1), 39-50.

Fornell, C., & Bookstein, F. L. (1982). Two Structural Equation Models: LISREL and PLS Applied to Consumer Exit-Voice Theory. *Journal of Marketing Research*, *19*(4), 440-452. doi: 10.2307/3151718

Fornell, C., & Cha, J. (1994). Partial Least Squares. In R. P. Bagozzi (Ed.), *Advanced Methods of Marketing Research* (pp. 52-78). Massachusetts: Blackwell.

Fowler, A., & Cusack, B. (2011). *Kodu game lab: improving the motivation for learning programming concepts.* Paper presented at the Proceedings of the 6th International Conference on Foundations of Digital Games, Bordeaux, France.

Fransson, A. (1977). On qualitative differences in learning. IV - Effects of motivation and test anxiety on process and outcome. *British Journal of Educational Psychology*, *47*, 244-257.

Fredricks, J. A., Blumenfeld, P. C., & Paris, A. H. (2004). School Engagement: Potential of the Concept, State of the Evidence. *Review of Educational Research*, *74*(1), 59-109. doi: 10.2307/1131219

Fredricks, J. A., Blumenfeld, P. C., Friedel, J., & Paris, A. H. (2005). School Engagement. In K. A. Moore & L. Lippman (Eds.), *Conceptualizing and measuring indicators of positive development: What do children need to flourish* (pp. 305-321). New York: Kluwer Academic/Plenum Press.

Fredricks, J. A., & McColskey, W. (2012). The Measurement of Student Engagement: A Comparative Analysis of Various Methods and Student Self-report Instruments. In S. L. Christenson, A. L. Reschly & C. Wylie (Eds.), Handbook of Research on Student Engagement. Dordrecht: Springer Science+Business Media New York. Retrieved from http://VUW.eblib.com/patron/FullRecord.aspx?p=884351.

Frenzel, A. C., Thrash, T. M., Pekrun, R., & Goetz, T. (2007). Achievement Emotions in Germany and China -- A Cross-Cultural Validation of the Academic Emotions Questionnaire-Mathematics. *Journal of Cross-Cultural Psychology*, *38*(3), 302-309. doi: http://dx.doi.org/10.1177/0022022107300276

Fuller, M. B., Wilson, M. A., & Tobin, R. M. (2010). The national survey of student engagement as a predictor of undergraduate GPA: a cross-sectional and longitudinal examination. *Assessment & Evaluation in Higher Education*, *36*(6), 735-748. doi: 10.1080/02602938.2010.488791

Furrer, C., & Skinner, E. (2003). Sense of relatedness as a factor in children's academic engagement and performance. *Journal of Educational Psychology, 95*(1), 148-162. doi: 10.1037/0022-0663.95.1.148

Galyon, C., Blondin, C., Yaw, J., Nalls, M., & Williams, R. (2012). The relationship of academic self-efficacy to class participation and exam performance. *Social Psychology of Education, 15*(2), 233-249. doi: 10.1007/s11218-011-9175-x

Gefen, D., Straub, D. W., & Boudreau, M.-C. (2000). Structural Equation Modeling And Regression: Guidelines For Research Practice. *Communications of the Association for Information Systems, 4*(7). Retrieved from http://aisel.aisnet.org/cais/vol4/iss1/7/

Geisser, S. (1974). A Predictive Approach to the Random Effects Model. *Biometrika, 61*, 101-107.

Glaser, B. G., & Strauss, A. L. (1967). *The discovery of grounded theory: Strategies for qualitative research.* New York: Aldine de Gruyter.

Glastra, F. J., Hake, B. J., & Schedler, P. E. (2004). Lifelong Learning as Transitional Learning. *Adult Education Quarterly, 54*(4), 291-307. doi: 10.1177/0741713604266143

Glorfeld, L. W., & Fowler, G. C. (1982). Validation of a model for predicting aptitude for introductory computing. *SIGCSE Bulletin, 14*(1), 140-143. doi: 10.1145/953051.801355

Gomes, A., & Mendes, A. J. (2007). *Learning to program - difficulties and solutions.* Paper presented at the International Conference on Engineering Education – ICEE 2007, Coimbra, Portugal.

Gravetter, F. J., & Wallnau, L. B. (2014). *Essentials of Statistics for the Behavioral Sciences* (8th ed.): Cengage Learning.

Greenbaum, T. L. (1993). *The handbook for focus group research.* New York: Lexington Books.

Gregor, S. (2006). The nature of theory in information systems. *MIS Quarterly, 30*(3), 611-642.

Gregor, S., & Klein, G. (2014). Eight Obstacles to Overcome in the Theory Testing Genre. *Journal of the Association for Information Systems,* 15(11), i - xix.

Guba, E. G., & Lincoln, Y. S. (1994). Competing paradigms in qualitative research. In N. K. Denzin & Y. S. Lincoln (Eds.), *Handbook of qualitative research* (4th ed., Vol. 2). London: Sage.

Guibert, N., Girard, P., & Guittet, L. (2004). *Example-based programming: a pertinent visual approach for learning to program.* Paper presented at the Proceedings of the working conference on Advanced Visual Interfaces, Gallipoli, Italy.

Guzdial, M., & Soloway, E. (2002). Teaching the Nintendo generation to program. *Communications of the ACM, 45*(4), 17-21. doi: 10.1145/505248.505261

Hair, J. F., Black, W. C., Babin, B. J., & Anderson, R. E. (2010). *Multivariate Data Analysis* (7th ed.). Upper Saddle River, NJ: Prentice Hall.

Hair, J. F., Ringle, C. M., & Sarstedt, M. (2011). PLS-SEM: Indeed a Silver Bullet. *Journal of Marketing Theory and Practice, 19*, 139-151.

Hair, J. F., Ringle, C. M., & Sarstedt, M. (2013). Partial Least Squares Structural Equation Modeling: Rigorous Applications, Better Results and Higher Acceptance. *Long Range Planning, 46*(1–2), 1-12. doi: http://dx.doi.org/10.1016/j.lrp.2013.01.001

Hair, J. F., Hult, G. T. M., Ringle, C. M., & Sarstedt, M. (2014). *A Primer On Partial Least Squares Structural Equation Modeling (PLS-SEM)*: SAGE Publications, Inc.

Hartmann, B., MacDougall, D., Brandt, J., & Klemmer, S. R. (2010). *What would other programmers do: suggesting solutions to error messages.* Paper presented at the Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Atlanta, Georgia, USA.

Hassinen, M., & Mäyrä, H. (2006). *Learning programming by programming: a case study.* Paper presented at the Proceedings of the 6th Baltic Sea Conference on Computing Education Research: Koli Calling 2006, Uppsala, Sweden.

Heider, F. (1958). *The psychology of interpersonal relations.* New York: Wiley.

Hektner, J. M., Schmidt, J. A., & Csikzentmihalyi, M. (2007). *Experience sampling method: Measuring the quality of everyday life.* Tousand Oaks, CA: Sage.

Henseler, J. (2007). A New and Simple Approach to Multi-group Analysis in Partial Least Squares Path Modeling. In H. Martens & T. Naes (Eds.), *Causalities explored by indirect observation: Proceedings of the 5th International Symposium on PLS and Related Methods (PLS'07)* (pp. 104-107). Oslo.

Henseler, J., Ringle, C. M., & Sinkovics, R. R. (2009). The use of partial least squares path modeling in international marketing. *New Challenges to International Marketing*, *20*, 277-319. Emerald Group Publishing Limited.

Henseler, J., & Chin, W. W. (2010). A Comparison of Approaches for the Analysis of Interaction Effects Between Latent Variables Using Partial Least Squares Path Modeling. *Structural Equation Modeling: A Multidisciplinary Journal, 17*(1), 82-109. doi: 10.1080/10705510903439003

Henseler, J., & Sarstedt, M. (2013). Goodness-of-fit indices for partial least squares path modeling. *Computational Statistics*, *28*(2), 565-580. doi: 10.1007/s00180-012-0317-1

Herbert, C. W. (2007). *An Introduction to Programming Using Alice.* Boston, MA: Thomson Course Technology.

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design Science in Information Systems Research. *MIS Quarterly*, *28*(1), 75-105.

Hinkin, T. R. (1998). A Brief Tutorial on the Development of Measures for Use in Survey Questionnaires. *Organizational Research Methods, 1*(1), 104-121. doi: 10.1177/109442819800100106

Hong, W., Chan, F. K. Y., Thong, J. Y. L., Chasalow, L. C., & Dhillon, G. (2014). A Framework and Guidelines for Context-Specific Theorizing in Information Systems Research. *Information Systems Research, 25*(1), 111-136. doi: doi:10.1287/isre.2013.0501

Hou, M., & Austin, T. (2007). *Use of CSCL tool in Java applications programming: a case study*. Paper presented at the Proceedings of the 45th annual southeast regional conference, Winston-Salem, North Carolina.

Hrastinski, S. (2009). A theory of online learning as online participation. *Computers & Education*, *52*(1), 78-82. doi: http://dx.doi.org/10.1016/j.compedu.2008.06.009

Hughes, J., & Peiris, D. R. (2006). *ASSISTing CS1 students to learn: learning approaches and object-oriented programming.* Paper presented at the Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education, Bologna, Italy.

Iivari, J., Hirschheim, R., & Klein, H. K. (1998). A Paradigmatic Analysis Contrasting Information Systems Development Approaches and Methodologies. *Information Systems Research*, *9*(2), 164-193. doi: 10.1287/isre.9.2.164

International Standard Classification of Education ISCED 2011. (2012) Retrieved 14 March, 2016, from http://www.uis.unesco.org/Education/Documents/isced-2011-en.pdf

Ippolito, I. (1997). PlanetSourceCode. Retrieved 23 May, 2013, from http://planetsourcecode.com

Jarvis, C. B., MacKenzie, S. B., & Podsakoff, P. M. (2003). A Critical Review of Construct Indicators and Measurement Model Misspecification in Marketing and Consumer Research. *Journal of Consumer Research*, *30*, 199-218.

Jenkins, T. (2002). *On the Difficulty of Learning to Program.* Paper presented at the 3rd Annual Conference of LTSN-ICS, Loughborough.

Jinwoo, K., & Lerch, F. J. (1997). Why is Programming (Sometimes) so Difficult? Programming as Scientific Discovery in Multiple Problem Spaces. *Information Systems Research*, *8*(1), 25.

Joo, Y. J., Lim, K. Y., & Kim, J. (2013). Locus of control, self-efficacy, and task value as predictors of learning outcome in an online university context. *Computers & Education*, *62*(0), 149-158. doi: http://dx.doi.org/10.1016/j.compedu.2012.10.027

Kahu, E. R. (2011). Framing student engagement in higher education. *Studies in Higher Education*, 1-16. doi: 10.1080/03075079.2011.598505

Karabenick, S. A. (2003). Seeking help in large college classes: A person-centered approach. *Contemporary Educational Psychology*, *28*(1), 37.

Karabenick, S. A., & Newman, R. S. (2006). *Help-seeking in academic settings: goals, groups, and contexts.* Mahwah, N.J: Lawrence Erlbaum Publishers.

Keil, M., Tan, B. C. Y., Wei, K.-K., Saarinen, T., Tuunainen, V., & Wassenaar, A. (2000). A Cross-Cultural Study on Escalation of Commitment Behavior in Software Projects. *MIS Quarterly*, *24*(2), 299-325. doi: 10.2307/3250940

Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Survey*, *37*(2), 83-137. doi: 10.1145/1089733.1089734

Kinnunen, P., & Malmi, L. (2006). *Why students drop out CS1 course?* Paper presented at the Proceedings of the second international workshop on Computing education research, Canterbury, United Kingdom.

Kline, P. (1999). *The Handbook of Psychological Testing* (2nd ed.). London: Routledge.

Ko, A. J., Abraham, R., Beckwith, L., Blackwell, A., Burnett, M., Erwig, M., . . . Wiedenbeck, S. (2011). The state of the art in end-user software engineering. *ACM Computing Survey, 43*(3), 1-44. doi: 10.1145/1922649.1922658

Kolb, A. Y., & Kolb, D. A. (2005). Learning Styles and Learning Spaces: Enhancing Experiential Learning in Higher Education. *Academy of Management Learning & Education*, *4*(2), 193-212.

Komarraju, M., & Nadler, D. (2013). Self-efficacy and academic achievement: Why do implicit beliefs, goals, and effort regulation matter? *Learning and Individual Differences*, *25*(0), 67-72. doi: http://dx.doi.org/10.1016/j.lindif.2013.01.005

Krueger, R. A., & Casey, M. A. (2009). *Focus groups: A Practical Guide for Applied Research* (4th ed.). Thousand Oaks, California: SAGE Publications, Inc.

Kuh, G. D., Cruce, T. M., Shoup, R., Kinzie, J., & Gonyea, R. M. (2008). Unmasking the Effects of Student Engagement on First-Year College Grades and Persistence. *The Journal of Higher Education 79*(5), 540-563 doi: 10.1353/jhe.0.0019

Kuh, G. D. (2009). The national survey of student engagement: Conceptual and empirical foundations. *New Directions for Institutional Research*, (141), 5-20. doi: 10.1002/ir.283

Lahtinen, E., Ala-Mutka, K., & Jarvinen, H.-M. (2005). A study of the difficulties of novice programmers. *SIGCSE Bulletin, 37*(3), 14-18. doi: 10.1145/1151954.1067453

Landis, J. R., & Koch, G. G. (1977). The Measurement of Observer Agreement for Categorical Data. *Biometrics, 33*(1), 159-174. doi: 10.2307/2529310

Langton, J. T., Hickey, T. J., & Alterman, R. (2004). Integrating tools and resources: a case study in building educational groupware for collaborative programming. *Journal of Computing in Small Colleges, 19*(5), 140-153.

Lee, M. J., & Ko, A. J. (2011). *Personifying programming tool feedback improves novice programmers' learning.* Paper presented at the Proceedings of the seventh international workshop on Computing education research, Providence, Rhode Island, USA.

Li, F. W. B., & Watson, C. (2011). *Game-based concept visualization for learning programming.* Paper presented at the Proceedings of the third international ACM workshop on Multimedia technologies for distance learning, Scottsdale, Arizona, USA.

Li, H., Xing, Z., Peng, X., & Zhao, W. (2013). *What help do developers seek, when and how?* Paper presented at the 2013 20th Working Conference on Reverse Engineering (WCRE)

Liebenberg, J., Mentz, E., & Breed, B. (2012). Pair programming and secondary school girls' enjoyment of programming and the subject Information Technology (IT). *Computer Science Education, 22*(3), 219-236. doi: 10.1080/08993408.2012.713180

Linnenbrink, E. A., & Pintrich, P. R. (2003). The role of self-efficacy beliefs in student engagement and learning in the classroom. *Reading & Writing Quarterly, 19*(2), 119-137. doi: 10.1080/10573560308223

Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., . . . Thomas, L. (2004). A multi-national study of reading and tracing skills in novice programmers. *SIGCSE Bulletin. 36*(4), 119-150.

Liu, K.-S., Cheng, Y.-Y., Chen, Y.-L., & Wu, Y.-Y. (2009). Longitudinal Effects Of Educational Expectation And Achievement Attributions On Adolescents' Academic Achievements. *Adolescence, 44*(176), 911-924.

Lohmöller, J. B. (1989). *Latent variable path modeling with partial least squares.* Heidelberg: Physica.

Lorsbach, A., & Jinks, J. (1999). Self-efficacy Theory and Learning Environment Research. *Learning Environments Research, 2*(2), 157-167. doi: 10.1023/a:1009902810926

MacCallum, R. C., Roznowski, M., Mar, C. M., & Reith, J. V. (1994). Alternative Strategies for Cross-Validation of Covariance Structure Models. *Multivariate Behavioral Research, 29*(1), 1-32. doi: 10.1207/s15327906mbr2901_1

MacCallum, R. C., Widaman, K. F., Zhang, S., & Hong, S. (1999). Sample Size in Factor Analysis. *Psychological Methods, 4*(1), 84-99.

MacKenzie, S. B., Podsakoff, P. M., & Podsakoff, N. P. (2011). Construct Measurement And Validation Procedures In MIS and Behavioral Research: Integrating New And Existing Techniques. *MIS Quarterly*, *35*(2), 293-A295.

MacKinnon, D. P., Lockwood, C. M., Hoffman, J. M., West, S. G., & Sheets, V. (2002). A comparison of methods to test mediation and other intervening variable effects. *Psychological Methods, 7*(1), 83-104. doi: 10.1037/1082-989x.7.1.83

MacKinnon, D. P. (2008). *Introduction to Statistical Mediation Analysis.* New York: Lawrence Erlbaum Associates.

Magnenat, S., Shin, J., Riedo, F., Siegwart, R., & Ben-Ari, M. (2014). *Teaching a core CS concept through robotics.* Paper presented at the Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education, Uppsala, Sweden.

Maguire, P., Maguire, R., Hyland, P., & Marshall, P. (2014). Enhancing collaborative learning using pair programming: Who benefits? *AISHE-J: The All Ireland Journal of Teaching and Learning in Higher Education 6*(2), 1411-14125.

Major, L., Kyriacou, T., & Brereton, P. (2014). The effectiveness of simulated robots for supporting the learning of introductory programming: a multi-case case study. *Computer Science Education, 24*(2-3), 193-228. doi: 10.1080/08993408.2014.963362

Maleko, M., Hamilton, M., & D'Souza, D. (2012). *Novices' perceptions and experiences of a mobile social learning environment for learning of programming.* Paper presented at the Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education, Haifa, Israel.

Mancy, R., & Reid, N. (2004). *Aspects of cognitive style and programming.* Paper presented at the Sixteenth Annual Workshop of the Psychology of Programming Interest Group (PPIG 2004), Carlow, Ireland.

Marton, F., & Säljö, R. (1976). On qualitative difference in learning. II - Outcome as a function of the learner's conception of the task. *British Journal of Educational Psychology, 46*, 115-127.

Marton, F., & Booth, S. (1997). *Learning and Awareness.* New Jersey: Lawrence Erlbaum Associates, Inc.

Mason, R., Cooper, G., & Raadt, M. d. (2012). *Trends in introductory programming courses in Australian universities: languages, environments and pedagogy.* Paper presented at the Proceedings of the Fourteenth Australasian Computing Education Conference - Volume 123, Melbourne, Australia.

Matzat, U., & Sadowski, B. (2012). Does the "Do-It-Yourself Approach" Reduce Digital Inequality? Evidence of Self-Learning of Digital Skills. *Information Society, 28*(1), 1-12. doi: 10.1080/01972243.2011.629023

McCaslin, M. M., & Good, T. L. (1996). Listening in classrooms. New York: HarperCollins.

McClure, J., Meyer, L. H., Garisch, J., Fischer, R., Weir, K. F., & Walkey, F. H. (2011). Students' attributions for their best and worst marks: Do they relate to achievement? *Contemporary Educational Psychology*, *36*(2), 71-81. doi: http://dx.doi.org/10.1016/j.cedpsych.2010.11.001

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., . . . Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *SIGCSE Bulletin, 33*(4), 125-180. doi: 10.1145/572139.572181

McKinney, D., & Denton, L. F. (2004). Houston, we have a problem: there's a leak in the CS1 affective oxygen tank. *SIGCSE Bulletin*, *36*(1), 236-239. doi: 10.1145/1028174.971386

Meyer, J. H. F., & Land, R. (2003). Threshold concepts and troublesome knowledge: Linkages to ways of thinking and practising within the disciplines. Retrieved from http://www.ed.ac.uk/etl/docs/ETLreport4.pdf

Meyer, J. H. F., & Land, R. (2006). Threshold concepts and troublesome knowledge: an introduction. In J. H. F. Meyer & R. Land (Eds.), *Overcoming Barriers to Student Understanding: Threshold Concepts and Troublesome Knowledge.* New York: Routledge.

Michael Page. (2015). [Chart illustrating occupations that are the most in demand across the world]. *The World's Most In Demand Professions*. Retrieved from http://www.michaelpage.com/minisite/most-in-demand-professions/

Miles, M. B., & Huberman, A. M. (1994). *Qualitative Data Analysis* (2nd ed.). Thousand Oaks, CA: SAGE.

Miles, M. B., Huberman, A. M., & Saldaña, J. (2014). Qualitative Data Analysis: A Methods Sourcebook (3rd ed.). California: SAGE Publications, Inc.

Miller, C. M. L., & Parlett, M. (1974). *Up to Mark: a study of the examination game*. London: Society for Research into Higher Education.

Miller, R. B., Greene, B. A., Montalvo, G. P., Ravindran, B., & Nichols, J. D. (1996). Engagement in academic work: The role of learning goals, future consequences, pleasing others, and perceived ability. *Contemporary Educational Psychology*, *21*(4), 388-422. doi: 10.1006/ceps.1996.0028

Mills, N., Pajares, F., & Herron, C. (2007). Self-efficacy of College Intermediate French Students: Relation to Achievement and Motivation. *Language Learning*, *57*(3), 417-442. doi: 10.1111/j.1467-9922.2007.00421.x

Milne, I., & Rowe, G. (2002). Difficulties in Learning and Teaching Programming--Views of Students and Tutors. *Education and Information Technologies*, *7*(1), 55-66. doi: 10.1023/a:1015362608943

Moore, G. C., & Benbasat, I. (1991). Development of an Instrument to Measure the Perceptions of Adopting an Information Technology Innovation. *Information Systems Research*, *2*(3), 192-222.

Moreno, A., Myller, N., & Sutinen, E. (2004). *JeCo, a Collaborative Learning Tool for Programming.* Paper presented at the Visual Languages and Human Centric Computing, 2004 IEEE Symposium on Visual Languages - Human Centric Computing (VLHCC'04).

Morgan, A. (1993). *Improving your Students' Learning*. London and Philadelphia: Kogan Page.

Morgan, D. L. (1997). Planning and Research Design for Focus Groups. In D. L. Morgan (Ed.), *Focus Groups as Qualitative Research* (pp. 32-46). Thousand Oaks, CA: SAGE Publications, Inc.

Mischel, W., Shoda, Y., & Rodriguez, M. (1989). Delay of gratification in children. *Science*, *244*(4907), 933-938. doi: 10.1126/science.2658056

Moskal, B., Lurie, D., & Cooper, S. (2004). Evaluating the effectiveness of a new instructional approach. *SIGCSE Bulletin*, *36*(1), 75-79. doi: 10.1145/1028174.971328

Murphy, C., Kim, E., Kaiser, G., & Cannon, A. (2008). Backstop: a tool for debugging runtime errors. *SIGCSE Bulletin, 40*(1), 173-177. doi: 10.1145/1352322.1352193

Murphy, C., Kaiser, G., Loveland, K., & Hasan, S. (2009). Retina: helping students and instructors based on observed programming activities. *SIGCSE Bulletin, 41*(1), 178-182. doi: 10.1145/1539024.1508929

Nagappan, N., Williams, L., Ferzli, M., Wiebe, E., Yang, K., Miller, C., & Balik, S. (2003). *Improving the CS1 experience with pair programming.* Paper presented at the Proceedings of the 34th SIGCSE technical symposium on Computer science education, Reno, Nevada, USA.

National Research Council and The Institute of Medicine. (2004). *Engaging Schools: Fostering high school students' motivation to learn.* Washington: The National Academic Press.

Nelson-Le Gall, S., & Glor-Scheib, S. (1985). Help seeking in elementary classrooms: An observational study. *Contemporary Educational Psychology, 10*(1), 58-71. doi: http://dx.doi.org/10.1016/0361-476X(85)90006-2

Nevalainen, S., & Sajaniemi, J. (2008). An Experiment on the Short-Term Effects of Engagement and Representation in Program Animation. *Journal of Educational Computing Research, 39*(4), 395-430. doi: 10.2190/EC.39.4.e

Nevins, D. (2013). CS 1 language adoption at California Community Colleges: 2012. *ACM Inroads, 4*(1), 34-37. doi: 10.1145/2432596.2432611

Newby, M., & Nguyen, T. H. (2010). Using the Same Problem with Different Techniques in Programming Assignments: An Empirical Study of its Effectiveness. *Journal of Information Systems Education, 21*(4), 375-382.

Newmann, F., Wehlage, G. G., & Lamborn, S. D. (1992). The significance and sources of student engagement. In F. Newmann (Ed.), *Student engagement and achievement in secondary schools* (pp. 11 - 39). New York: Teachers College Press.

Nunnally, J. C. (1978). *Psychometric theory*: McGraw-Hill.

Nunnally, J. C., & Bernstein, I. (1994). *Psychometric Theory* (3rd ed.). New York: McGraw-Hill.

O'Keefe, P. A., & Linnenbrink-Garcia, L. (2014). The role of interest in optimizing performance and self-regulation. *Journal of Experimental Social Psychology, 53*(0), 70-78. doi: http://dx.doi.org/10.1016/j.jesp.2014.02.004

Orlikowski, W. J., & Baroudi, J. J. (1991). Studying Information Technology in Organizations: Research Approaches and Assumptions. *Information Systems Research, 2*(1), 1-28.

Pair, C. (1993). Programming, Programming Languages and Programming Methods. In J. M. Hoc, T. R. G. Green, R. Samurcay & F. J. Gilmore (Eds.), *Psychology of Programming* (pp. 9-19). New York: Academic Press.

Pajares, F. (1997). Current Directions in self-efficacy research. In M.Maehr & P. R. Pintrich (Eds.), Advances in motivation and achievement (Vol. 10, pp. 1 - 49). Greenwich, CT: JAI Press.

Pekrun, R., Goetz, T., Titz, W., & Perry, R. P. (2002). Academic Emotions in Students' Self-Regulated Learning and Achievement: A Program of Qualitative and Quantitative Research. *Educational Psychologist, 37*(2), 91-105. doi: 10.1207/s15326985ep3702_4

Pekrun, R., Goetz, T., Perry, R. P., Kramer, K., Hochstadt, M., & Molfenter, S. (2004). Beyond test anxiety: Development and validation of the test emotions questionnaire (TEQ). *Anxiety, Stress, & Coping*, *17*(3), 287-316. doi: 10.1080/10615800412331303847

Phan, H. P. (2011). Interrelations between self-efficacy and learning approaches: a developmental approach. *Educational Psychology*, *31*(2), 225-246. doi: 10.1080/01443410.2010.545050

Pinsonneault, A., & Kraemer, K. L. (1993). Survey Research Methodology in Management Information Systems: An Assessment. *Journal of Management Information Systems*, *10*(2), 75-105. doi: 10.2307/40398056

Pintrich, P. R., & de Groot, E. V. (1990). Motivational and self-regulated learning components of classroom academic performance. *Journal of Educational Psychology*, *82*(1), 33-40. doi: 10.1037/0022-0663.82.1.33

Pintrich, P. R., & Schunk, D. H. (1996). *Motivation in education: Theory, research and applications* (2nd ed.). Upper Saddle River, NJ: Merrill Prentice Hall

Podsakoff, P. M., & Organ, D. W. (1986). Self-reports in Organizational Research: Problems and Prospects. *Journal of Management*, *12*, 531-544.

Podsakoff, P. M., MacKenzie, S. B., Lee, J.-Y., & Podsakoff, N. P. (2003). Common method biases in behavioral research: a critical review of the literature and recommended remedies. *Journal of Applied Psychology*, *88*(5), 879-903.

Poropat, A. E. (2009). A meta-analysis of the five-factor model of personality and academic performance. *Psychological Bulletin*, *135*(2), 322-338.

Powers, K., Gross, P., Cooper, S., McNally, M., Goldman, K. J., Proulx, V., & Carlisle, M. (2006). Tools for teaching introductory programming: what works? *SIGCSE Bulletin, 38*(1), 560-561. doi: 10.1145/1124706.1121514

PPIG.org. (n.d.). Psychology of Programming Interest Group. Retrieved 10 February 2013, from http://ppig.org/

Preacher, K., & Hayes, A. (2008). Asymptotic and resampling strategies for assessing and comparing indirect effects in multiple mediator models. *Behavior Research Methods*, *40*(3), 879-891. doi: 10.3758/brm.40.3.879

Punch, K. (2005). *Introduction to Social Research: Quantitative and Qualitative Approaches* (2nd ed.). London: SAGE Publications.

Putwain, D., Sander, P., & Larkin, D. (2013). Academic self-efficacy in study-related skills and behaviours: Relations with learning-related emotions and academic success. *British Journal of Educational Psychology*, *83*(4), 633-650. doi: 10.1111/j.2044-8279.2012.02084.x

Qualtrics, Provo, UT Qualtrics Research Suite (Version 59038 0.698s (0.336, 0.265, 0.049, 0.153, 0.018)) [Software]. (2005). Copyright © 2014 Qualtrics. Qualtrics and all other Qualtrics product or service names are registered trademarks or trademarks of Qualtrics, Provo, UT, USA. Retrieved 15 March 2015, from http://www.qualtrics.com.

Raccoon, L. B. S. (1997). Fifty years of progress in software engineering. *SIGSOFT Software Engineering Notes, 22*(1), 88-104. doi: 10.1145/251759.251878

Rajkumar, T. M., Anderson, P., Benamati, J., & Merhout, J. W. (2011). Are Student Self-Assessments a Valid Proxy for Direct Assessments in Efforts to Improve Information Systems Courses and Programs? An Empirical Study. *Communications of AIS, 28*, 537-548.

Ramalingam, V., & Wiedenbeck, S. (1998). Development and Validation of Scores on a Computer Programming Self-Efficacy Scale and Group Analyses of Novice Programmer Self-Efficacy. *Journal of Educational Computing Research, 19*(4), pp 367 - 381 doi: 10.2190/C670-Y3C8-LTJ1-CT3P

Ramalingam, V., LaBelle, D., & Wiedenbeck, S. (2004). Self-efficacy and mental models in learning to program. *SIGCSE Bulletin*, *36*(3), 171-175. doi: 10.1145/1026487.1008042

Ramsden, P. (1979). Student learning and perceptions of the academic environment. *Higher Education*, *8*, 411-428.

Ramsden, P. (2003). *Learning to teach in higher education* (2nd ed.). GB: Routledge Falmer.

Reschly, A. L., & Christenson, S. L. (2012). Jingle, Jangle, and Conceptual Haziness: Evolution and Future Directions of the Engagement Construct. In S. L. Christenson, A. L. Reschly & C. R. f. Wylie (Eds.), Handbook of Research on Student Engagement. Dordrecht: Springer Science+Business Media New York. Retrieved from http://VUW.eblib.com/patron/FullRecord.aspx?p=884351.

Resnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., . . . Kafai, Y. (2009). Scratch: programming for all. *Communications of the ACM*, *52*(11), 60-67. doi: 10.1145/1592761.1592779

Ringle, C. M., Sarstedt, M., & Straub, D. W. (2012). A Critical Look at the Use of PLS-SEM in MIS Quarterly. *MIS Quarterly*, *36*(1), iiv-8.

Ringle, C. M., Wende, S., & Becker, J-M. (2014). *SmartPLS 3*. Hamburg: SmartPLS. Retrieved from http://www.smartpls.com

Roberts, N., & Grover, V. (2009). Handbook of Research on Contemporary Theoretical Models in Information Systems. In Y. Dwivedi, B. Lal, M. Williams, S. Schneberger & M. Wade (Eds.), *Theory Development in Information Systems Research Using Structural Equation Modeling: Evaluation and Recommendations* (pp. 77-94). Hershey, PA: Information Science Reference.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. *Computer Science Education, 13*(2), 137-172. doi: 10.1076/csed.13.2.137.14200

Rodrigo, M. M. T., Baker, R. S., Jadud, M. C., Amarra, A. C. M., Dy, T., Espejo-Lahoz, M. B. V., . . . Tabanao, E. S. (2009). Affective and behavioral predictors of novice programmer achievement. *SIGCSE Bulletin*, *41*(3), 156-160. doi: 10.1145/1595496.1562929

Rogers, C. R. (1951). *Client-centered therapy: Its current practice, implications, and theory*. Boston: Houghton, Mifflin.

Rönkkö, M., & Evermann, J. (2013). A critical examination of common beliefs about partial least squares path modeling. *Organizational Research Methods, 16*(3), 425-448. doi: http://dx.doi.org/10.1605/01.301-0022978611.2013

Rotter, J. B. (1966). Generalized expectancies for internal versus external control of reinforcement. *Psychological Monographs, 80,* 148–154.

Rottinghaus, P. J., Larson, L. M., & Borgen, F. H. (2003). The relation of self-efficacy and interests: a meta-analysis of 60 samples. *Journal of Vocational Behavior, 62*(2), 221-236. doi: http://dx.doi.org/10.1016/S0001-8791(02)00039-8

Rountree, N., Rountree, J., & Robins, A. (2002). Predictors of success and failure in a CS1 course. *SIGCSE Bulletin, 34*(4), 121-124. doi: 10.1145/820127.820182

Rountree, J., & Rountree, N. (2009). *Issues regarding threshold concepts in computer science.* Paper presented at the Proceedings of the Eleventh Australasian Conference on Computing Education - Volume 95, Wellington, New Zealand.

Rubio, M. A., Romero-Zaliz, R., Mañoso, C., & de Madrid, A. P. (2015). Closing the gender gap in an introductory programming course. *Computers & Education, 82*(0), 409-420. doi: http://dx.doi.org/10.1016/j.compedu.2014.12.003

Ryan, A. M., & Pintrich, P. R. (1998). Achievement and social motivational influences on help seeking in the classroom. In S. A. Karabenick (Ed.), *Strategic help seeking: Implications for learning and teaching* (pp. 117-139). Mahwah, NJ: Erlbaum.

Ryan, A. M., & Shin, H. (2011). Help-seeking tendencies during early adolescence: An examination of motivational correlates and consequences for achievement. *Learning and Instruction, 21*(2), 247-256. doi: http://dx.doi.org/10.1016/j.learninstruc.2010.07.003

Samurcay, R. (1989). The concept of variable in programming: Its meaning and use in problem-solving by novice programmers. In E. Soloway & J. C. Spohrer (Eds.), *Studying the Novice Programmer* (pp. 161-178). Hilsdale, NJ: Lawrence Erlbaum Associates.

Sarstedt, M., Henseler, J., & Ringle, C. M. (2011). Multi-group Analysis in Partial Least Squares (PLS) Path Modeling: Alternative Methods and Empirical Results. *Advances in International Marketing, 22*, 195-218.

Schreiber, J. B., Nora, A., Stage, F. K., Barlow, E. A., & King, J. (2006). Reporting Structural Equation Modeling and Confirmatory Factor Analysis Results: A Review. *The Journal of Educational Research, 99*(6), 323-327,330-337,384.

Schunk, D. H. (1991). Self-Efficacy and Academic Motivation. *Educational Psychologist, 26*(3/4), 207.

Schunk, D. H. (1996). *Self-Efficacy for Learning and Performance.* Paper presented at the Annual Meeting of the American Educational Research Association, New York.

Schunk, D. H., Pintrich, P. R., & Meece, J. L. (2008). *Motivation in Education: Theory, Research and Applications* (3rd ed.). Upper Saddle River, NJ: Pearson Education.

Schunk, D. H., & Mullen, C. A. (2012). Self-Efficacy as an Engaged Learner. In S. L. Christenson, A. L. Reschly & C. Wylie (Eds.), Handbook of Research on Student Engagement. Dordrecht: Springer Science+Business Media New York. Retrieved from http://VUW.eblib.com/patron/FullRecord.aspx?p=884351.

Shaw, R.S. (2013). The relationships among group size, participation, and performance of programming language learning supported with online forums. *Computers & Education, 62*, 196-207. doi: 10.1016/j.compedu.2012.11.001

Sheard, J., & Hagan, D. (1998). Our failing students: a study of a repeat group. *SIGCSE Bulletin, 30*(3), 223-227. doi: 10.1145/290320.283550

Sheard, J., Carbone, A., & Hurst, A. J. (2010). Student engagement in first year of an ICT degree: staff and student perceptions. *Computer Science Education, 20*(1), 1-16. doi: 10.1080/08993400903484396

Sherer, M., Maddux, J. E., Mercandante, B., Prentice-Dunn, S., Jacobs, B., & Rogers, R. W. (1982). The Self-Efficacy Scale: Construction and Validation. *Psychological Reports*, *51*(2), 663-671. doi: 10.2466/pr0.1982.51.2.663

Shi, S. G. (1992). Accurate and efficient double-bootstrap confidence limit method. *Computational Statistics & Data Analysis*, *13*(1), 21-32. doi: http://dx.doi.org/10.1016/0167-9473(92)90151-5

Shih, H.-P. (2006). Assessing the effects of self-efficacy and competence on individual satisfaction with computer use: an IT student perspective. *Computers in Human Behavior*, *22*(6), 1012-1026. doi: 10.1016/j.chb.2004.03.025

Shinners-Kennedy, D. (2008). The Everydayness of Threshold Concepts: 'State' as an Example from Computer Science. In R. Land, J. H. F. Meyer & J. Smith (Eds.), *Threshold Concepts within the Disciplines.* Rotterdam: Sense Publishers.

Sillitti, A., Succi, G., & Vlasenko, J. (2012). *Understanding the impact of pair programming on developers attention: a case study on a large industrial experimentation.* Paper presented at the Proceedings of the 2012 International Conference on Software Engineering, Zurich, Switzerland.

Silvia, P. J. (2003). Self-efficacy and interest: Experimental studies of optimal incompetence. *Journal of Vocational Behavior*, *62*(2), 237-249. doi: http://dx.doi.org/10.1016/S0001-8791(02)00013-1

Simon, Fincher, S., Robins, A., Baker, B., Box, I., Cutts, Q., . . . Tutty, J. (2006). *Predictors of success in a first programming course.* Paper presented at the Proceedings of the 8th Australasian Conference on Computing Education - Volume 52, Hobart, Australia.

Skinner, E., Furrer, C., Marchand, G., & Kindermann, T. (2008). Engagement and disaffection in the classroom: Part of a larger motivational dynamic? *Journal of Educational Psychology*, *100*(4), 765-781. doi: 10.1037/0033-2909.117.3.497

Skinner, E. A., Kindermann, T. A., Connell, J. P., & Wellborn, J. G. (2009). Engagement and disaffection as organizational constructs in the dynamics of motivational development. In K. R. Wentzel & A. Wigfield (Eds.), *Handbook of Motivation at School* (pp. pp. 223 - 245). New York: Routledge.

Skinner, E. A., Kindermann, T. A., & Furrer, C. J. (2009). A Motivational Perspective on Engagement and Disaffection: Conceptualization and Assessment of Children's Behavioral and Emotional Participation in Academic Activities in the Classroom. *Educational and Psychological Measurement*, *69*(3), 493-525. doi: 10.1177/0013164408323233

Skinner, E. A., & Pitzer, J. R. (2012). Developmental dynamics of student engagement, coping, and everyday resilience. In S. L. Christenson, A. L. Reschly & C. Wylie (Eds.), *Handbook of research on student engagement* (pp. 21-44). New York, NY: Springer.

Sobel, M. E. (1982). Asymptotic Confidence Intervals for Indirect Effects in Structural Equation Models. *Sociological Methodology, 13*, 290-312. doi: 10.2307/270723

Soloway, E., & Spohrer, J. C. (Eds.). (1989). *Studying the novice programmer*. Hilsdale, NJ: Lawrence Erlbaum.

Sorva, J. (2010). *Reflections on threshold concepts in computer programming and beyond*. Paper presented at the Proceedings of the 10th Koli Calling International Conference on Computing Education Research, Koli, Finland.

Stachel, J., Marghitu, D., Brahim, T. B., Sims, R., Reynolds, L., & Czelusniak, V. (2013). Managing Cognitive Load in Introductory Programming Courses: A Cognitive Aware Scaffolding Tool. *Journal of Integrated Design & Process Science, 17*(1), 37-54. doi: 10.3233/jid-2013-0004

Stangor, C. (2011). *Research Methods for the Behavioral Sciences.* Belmont, CA, USA: Wadsworth Cengage Learning.

Stewart, D. W., Shamdasani, P. N., & Rook, D. W. (2007). *Introduction: Focus Group History, Theory, and Practice.* Thousand Oaks, CA: SAGE Publications, Ltd.

Straub, D. W. (1989). Validating Instruments in MIS Research. *MIS Quarterly, 13*(2), 147-169. doi: 10.2307/248922

Straub, D., Boudreau, M.-C., & Gefen, D. (2004). Validation Guidelines for IS Positivist Research. *Communications of AIS, 13*, 380-427.

Strauss, A., & Corbin, J. (1990). *Basics of qualitative research Grounded theory procedures and techniques.* Newbury Park: Sage Publications.

Stone, M. (1974). Cross-validatory Choice and Assessment of Statistical Predictions. *Journal of the Royal Statistical Society*, 36, 111-147.

Svensson, L. (1977). On qualitative difference in learning. III - Study skill and learning. *British Journal of Educational Psychology, 47*, 233-243.

Teague, D., & Roe, P. (2008). *Collaborative learning: towards a solution for novice programmers.* Paper presented at the Proceedings of the tenth conference on Australasian computing education - Volume 78, Wollongong, NSW, Australia.

Teague, D. M., & Roe, P. (2009). *Learning to program: from pear-shaped to pairs.* Paper presented at the Proceedings of the First International Conference on Computer Supported Education, Lisboa, Portugal.

Teddlie, C., & Tashakkori, A. (2009). *Foundations of Mixed Methods Research.* Thousand Oaks, CA: Sage Publications.

Thomas, D. R. (2006). A General Inductive Approach for Analyzing Qualitative Evaluation Data. *American Journal of Evaluation, 27*(2), 237-246. doi: 10.1177/1098214005283748

Thuné, M., & Eckerdal, A. (2009). Variation theory applied to students' conceptions of computer programming. *European Journal of Engineering Education, 34*(4), 339-347. doi: 10.1080/03043790902989374

Tillmann, N., Halleux, J. D., Xie, T., Gulwani, S., & Bishop, J. (2013). *Teaching and learning programming and software engineering via interactive gaming.* Paper presented at the Proceedings of the 2013 International Conference on Software Engineering, San Francisco, CA, USA.

Tuchinda, R., Knoblock, C. A., & Szekely, P. (2011). Building Mashups by Demonstration. *ACM Transactions on the Web, 5*(3), 1-45. doi: 10.1145/1993053.1993058

UCAS. (2014). International Qualifications: For Entry to University of College in 2015 Retrieved 12 February 2015, from http://www.ucas.com/sites/default/files/2015-international-qualifications.pdf

Utting, I., Cooper, S., Kolling, M., Maloney, J., & Resnick, M. (2010). Alice, Greenfoot, and Scratch -- A Discussion. *ACM Transactions on Computing Education, 10*(4), 1-11. doi: 10.1145/1868358.1868364

van Teijlingen, E., & Hundley, V. (2001). The importance of pilot studies. *Social Research Update, 35*.

Venkatesh, V., Brown, S. A., & Bala, H. (2013). Bridging the qualitative-quantitative divide: Guidelines for conduction mixed methods research in Information Systems. *MIS Quarterly, 37*(1), 21-54.

Ventura, P. R. (2005). Identifying predictors of success for an objects-first CS1. *Computer Science Education, 15*(3), 223-243. doi: 10.1080/08993400500224419

Villalobos, J. A., Calderón, N. A., & Jiménez, C. H. (2009). Developing programming skills by using interactive learning objects. *SIGCSE Bulletin, 41*(3), 151-155. doi: 10.1145/1595496.1562927

von Stumm, S., Hell, B., & Chamorro-Premuzic, T. (2011). The Hungry Mind: Intellectual Curiosity Is the Third Pillar of Academic Performance. *Perspectives on Psychological Science, 6*(6), 574-588. doi: 10.1177/1745691611421204

Walker, C. O., Greene, B. A., & Mansell, R. A. (2006). Identification with academics, intrinsic/extrinsic motivation, and self-efficacy as predictors of cognitive engagement. *Learning and Individual Differences, 16*(1), 1-12. doi: http://dx.doi.org/10.1016/j.lindif.2005.06.004

Watson, C., & Li, F. W. B. (2014). *Failure rates in introductory programming revisited.* Paper presented at the Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education, Uppsala, Sweden.

Watson, C., Li, F. W. B., & Godwin, J. L. (2014). *No tests required: comparing traditional and dynamic predictors of programming success.* Paper presented at the Proceedings of the 45th ACM Technical Symposium on Computer Science Education, Atlanta, Georgia, USA.

Weber, R. (2012). Evaluating and Developing Theories in the Information Systems Discipline. *Journal of the Association for Information Systems, 13*(1), 1-30.

Weinberg, G. M. (1971). *The Psychology of Computer Programming.* New York: Van Nostrand Reinhold Company.

Weiner, B. (1985). An attributional theory of achievement motivation and emotion. *Psychological Review, 92*(4), 548-573. doi: http://dx.doi.org/10.1037/0033-295X.92.4.548

Wenger, E. (1998). *Communities of practice: Learning, meaning, and identity.* Cambridge: Cambridge University Press.

Whalley, J. L., Lister, R., Thompson, E., Clear, T., Robbins, P., Kumar, P. K. A., & Prasad, C. (2006). *An Australasian study of reading and comprehension skills in novice programmers, using the bloom and SOLO taxonomies.* Paper presented at the Proceedings of the 8th Australasian Conference on Computing Education - Volume 52, Hobart, Australia.

Whalley, J. L., & Lister, R. (2009). *The BRACElet 2009.1 (Wellington) specification.* Paper presented at the Proceedings of the Eleventh Australasian Conference on Computing Education - Volume 95, Wellington, New Zealand.

White, A. R. (1964). The Notion of Interest. *The Philosophical Quarterly, 14*(57), 319-327. doi: 10.2307/2217771

White, R. (2004). *Predicting the persistence of information technology students.* 3141905 Ph.D., University of Miami, Ann Arbor. Retrieved from http://search.proquest.com/docview/305179372?accountid=14782

White, G., & Sivitanides, M. (2002). A theory of the relationships between cognitive requirements of computer programming languages and programmers' cognitive characteristics. *Journal of Information Systems Education, 13*(1), 60-66.

White, G., & Ploeger, F. (2004). Cognitive characteristics for learning Visual Basic. *Journal of Computer Information Systems, 44*(3), 58-66.

Wiedenbeck, S. (2005). *Factors affecting the success of non-majors in learning to program.* Paper presented at the Proceedings of the first international workshop on Computing education research, Seattle, WA, USA.

Wiedenbeck, S., Xiaoning, S., & Chintakovid, T. (2007). *Antecedents to End Users' Success in Learning to Program in an Introductory Programming Course.* Paper presented at the Visual Languages and Human-Centric Computing, 2007. VL/HCC 2007. IEEE Symposium on.

Wigfield, A. (1994). Expectancy-value theory of achievement motivation: A developmental perspective. *Educational Psychology Review, 6*(1), 49-78. doi: 10.1007/bf02209024

Wilson, B. C., & Shrock, S. (2001). Contributing to success in an introductory computer science course: a study of twelve factors. *SIGCSE Bulletin, 33*(1), 184-188. doi: 10.1145/366413.364581

Winslow, L. E. (1996). Programming pedagogy - A psychological overview. *SIGCSE Bulletin, 28*(3), 17-22. doi: 10.1145/234867.234872

Wolfe, J. M. (1977). An interim validation report on the Wolfe: programming aptitude test (experimental form S). *SIGCPR Computer Personnel Research, 6*(1-2), 9-11. doi: 10.1145/1036942.1036944

Wood, R., & Bandura, A. (1989). Social Cognitive Theory of Organizational Management. *Academy of Management Review, 14*(3), 361-384. doi: 10.5465/amr.1989.4279067

Yazzie-Mintz, E., & McCormick, K. (2012). Finding the Humanity in the Data: Understanding, Measuring and Strengthening Student Engagement. In S. L. Christenson, A. L. Reschly & C. Wylie (Eds.), *Handbook of Research on Student Engagement* (pp. 743-762). Dordrecht: Springer Science+Business Media New York. Retrieved from http://VUW.eblib.com/patron/FullRecord.aspx?p=884351.

Yip, M. C. W. (2012). Learning strategies and self-efficacy as predictors of academic performance: a preliminary study. *Quality in Higher Education, 18*(1), 23-34. doi: 10.1080/13538322.2012.667263

Young, H. P. (2009). Learning by trial and error. *Games and Economic Behavior, 65*(2), 626-643. doi: http://dx.doi.org/10.1016/j.geb.2008.02.011

Zander, C., Boustedt, J., Eckerdal, A., McCartney, R., Mostrom, J. E., Ratcliffe, M., & Sanders, K. (2008). Threshold Concepts in Computer Science: A multi-national empirical investigation. In R. Land, J. H. F. Meyer & J. Smith (Eds.), *Threshold Concepts within the Disciplines* (pp. 105-118). Rotterdam: Sense Publishers.

Zander, C., Thomas, L., Simon, B., Murphy, L., McCauley, R., Hanks, B., & Fitzgerald, S. (2009). Learning styles: novices decide. *SIGCSE Bulletin, 41*(3), 223-227. doi: 10.1145/1595496.1562948

Zang, N., Rosson, M. B., & Nasser, V. (2008). *Mashups: who? what? why?* Paper presented at the CHI '08 Extended Abstracts on Human Factors in Computing Systems, Florence, Italy.

Zikmund, W. G., Babin, B. J., Carr, J. C., & Griffin, M. (2010). *Business Research Methods* (8th ed.). Mason, OH: South-Western, Cengage Learning.

Zimmerman, B. J. (1995). Self-efficacy and educational development. In A. Bandura (Ed.), *Self-efficacy in changing societies* (pp. pp. 202-231). New York: Cambridge University Press.

Zimmerman, B. J., & Risemberg, R. (1997). Self-regulatory dimensions of academic learning and motivation. In G. D. Phye (Ed.), *Handbook of academic learning: Construction of knowledge* (pp. 105–125). New York: Academic Press.

Zimmerman, B. J. (2000). Self-Efficacy: An Essential Motive to Learn. *Contemporary Educational Psychology*, *25*(1), 82-91. doi: 10.1006/ceps.1999.1016

Appendix A: Human Ethics Committee Approval Letter

VICTORIA
UNIVERSITY OF WELLINGTON

SCHOOL OF INFORMATION MANAGEMENT
TE KURA TIKKI, WHAKAWHITI KŌREPO
LEVEL 5, RUTHERFORD HOUSE, PIPITEA CAMPUS, 23 LAMBTON QUAY, WELLINGTON PO Box 600, Wellington 6140, New Zealand
Phone +64-4-463 5108 Fax +64-4-463 5416 Email sim@vuw.ac.nz
www.victoria.ac.nz/sim

28 October 2013

Geetha Kanaparan
c/- School of Information Management
Victoria University of Wellington
PO Box 600
Wellington 6140
NEW ZEALAND

Dear Geetha,

HUMAN ETHICS APPLICATION

Your Human Ethics application for Research Project: Self-efficacy and Engagement as Predictors of Student Programming, has been reviewed and the Committee's decision is the following:

Application accepted.

You may begin your data collection immediately.

Yours sincerely.

Dr David Johnstone
Chair, SIM Human Ethics Committee
School of Information Management

Appendix B: Analysis of Focus Groups

Table AB.1: Focus Groups Coding Table

Example of Coding				1st level of code	2nd level of code (Indicators of Engagement)
Focus Group 1 New Zealand	Focus Group 2 New Zealand	Focus Group 3 New Zealand	Focus Group 4 Malaysia		
"I pretty much only go into [the lecture] out of obligation.... I guess whatever is on the slides you can just pretty much read... if I miss a verbal snippet [lecture] I can just again Google it" (CW, FG1)	"attend lectures, which in a way handy and in a way not" (DI, 4) "I go to lectures, then I do the workshops..." (DI, FG2)	"I try to go [to lectures] but I feel guilty if I don't go. But I do tend to zone out a lot." (RB, FG3)		• Attend lectures • Attend workshops (practical) • Revise course notes • Seek help on discussion boards	Participation
"I ask tutor a lot" (SZ, FG1)	"... for other courses I can just have a look at the textbook and see if there is a resource and then talk to people. But for coding ... I talk with my friends before I start it because I don't know what's going on." (ZX, FG2)	"I use the lecture slides and I also Google lots of things." (RB, FG3)	"We all have that mentality where we go... let's try troubleshooting this.... Nope... Miss Miss....[call the tutor]" (MJE, FG4)	Seek help from: • Peer • Instructor • tutor • lecture slides • previous workshops • family members • Internet	Help-seeking

Example of Coding				1st level of code	2nd level of code (Indicators of Engagement)
Focus Group 1 New Zealand	Focus Group 2 New Zealand	Focus Group 3 New Zealand	Focus Group 4 Malaysia		
"...there is the assignment and then there are exercises for each part of the assignment. I do all the exercises...in the lab" (MH, FG1)	"I need exact examples...then I need to figure out what each part of it is doing...that piece results in that happening... [then] generalize it to the rest...figure out the logic..." (CL, FG2)		"You have to first try it yourself first. Programming is different. If other modules, you just have to read and memorize. But for programming, you need to understand. I try coding myself. I would go online and find some questions to follow. Look for examples and follow and see the result." (LT, FG4)	• Attempt lab exercises • Analyse programming code • Set time to revise • Understand code • Take notes • Practise programming	Effort
"..You understand it first, memorize it, you write it on paper again and again, practise a lot. Once you practise 100 times, you remember everything. You understand everything." (YL, FG1)		"Determination. It's like learning a whole new language." (JR, FG3)	"... very determined. You should not give up too soon." (FA, FG4)	• Practise programming • Determined • Resourceful • Keep trying	Persistence

Example of Coding				1st level of code	2nd level of code (Indicators of Engagement)
Focus Group 1 New Zealand	Focus Group 2 New Zealand	Focus Group 3 New Zealand	Focus Group 4 Malaysia		
"..sort of found there are 2 types of problems you can get.. trick is to learn to read things" (CM, FG1)	"After I have the initial understanding and the basis for everything, the best way for me is to teach others, explain it to others..., I didn't understand it [the assignment] until I was explaining it to like 3 different people..." (CL, FG2) "..you need to go away and try and think about what you need to putso you can say if this thing is selected, what happens then." (RS, FG2)	"...as far as logic errors go... print out the code and step through it with a pen and paper" (CM, FG3) "I think that preparing [for practical] would be better cause then I will know where everything is." (JR, FG3)	"I think my preparation before the lab session is during the lecture. I will make some notes. I will try to finish everything during the lecture so that I don't have to think about it after the lecture. During the lab session I will go back to the notes." (LT, FG4)	• Explain to peers • Analytical mind • Revise course notes • Prepare for classes • Take notes • Practise programming • Analyse code • Understand program logic • Step through code	Deep Learning
"spend like 3-5 hours per day on the weekend, just to memorize those codes and understand that code" (YL, FG1)			"We just go through the slides that we studied in class at that moment [during the practical session]..." (FA, FG4)	• Did not prepare for class • Rely on exam tips • Memorize code	Surface Learning

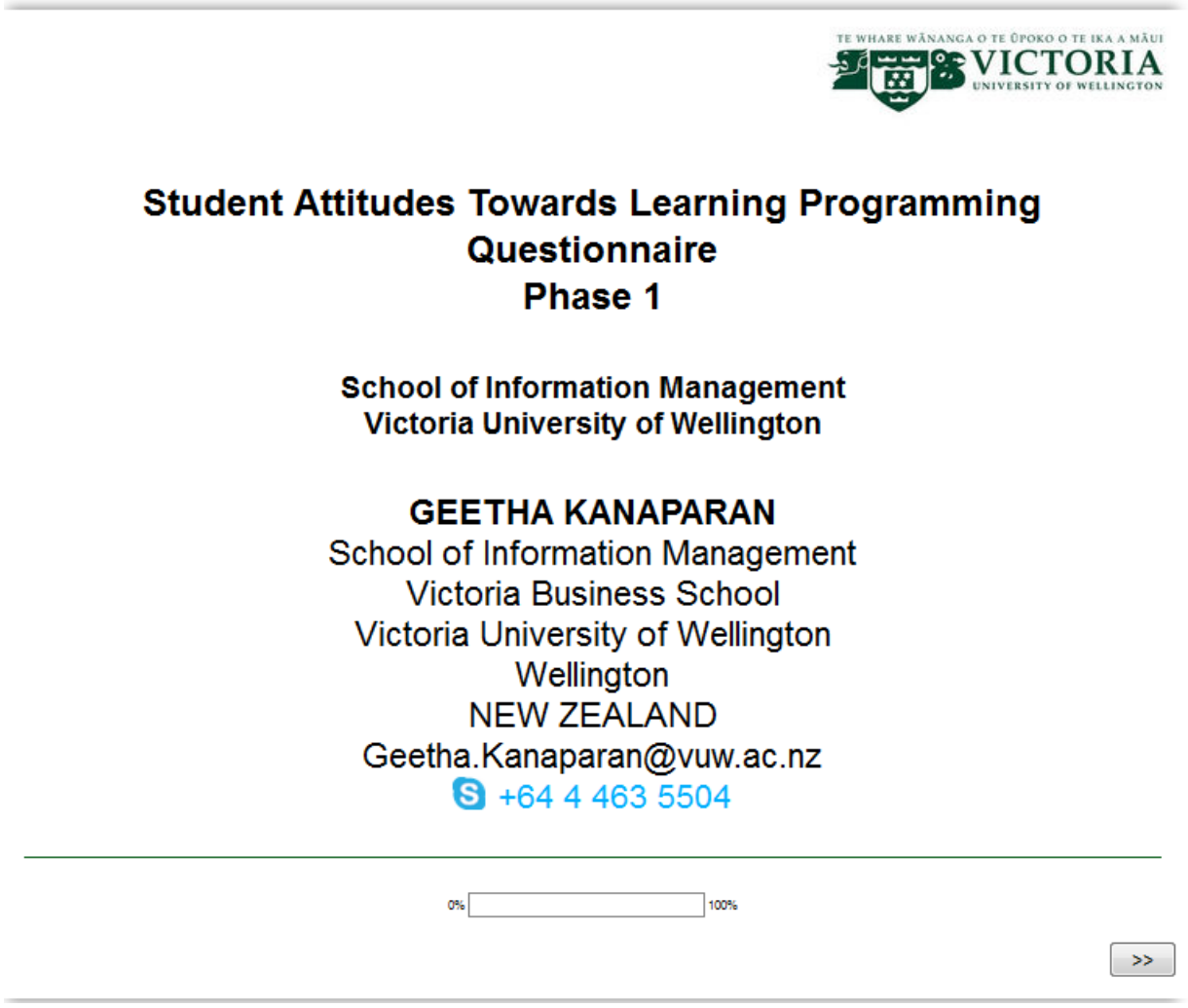
Example of Coding				1st level of code	2nd level of code (Indicators of Engagement)
Focus Group 1 New Zealand	Focus Group 2 New Zealand	Focus Group 3 New Zealand	Focus Group 4 Malaysia		
	"... I use previous workshops... I'll put this switch statement here, it works. I'll copy it into here or I'll move it into here and then I'll just fill in the empty gaps to make it relevant to this question." (DI, FG2)	"When I finally figured out it was that [why it wasn't working]; but I was changing everything else." (JR, FG3) "Just like go back to it and look through it. Sometimes you just have to really look at some code. Like put this line before that one, change a little things." (AL, FG3)		• Insert code that looks right • Move code around • Mess up codes to find solution • Compare codes • Code causing the error not obvious from error messages	Trial and Error
"I want to get good grades.. I make sure my code part and completion plus together over 80%" (SZ, FG1) "..my aim is I get good grades so I use the most efficient way to make my grade look great. But if I really know how to code, I don't know." (SZ, FG1)				• Focus on expected returns • Review past papers • Review past answers • Time proportionate to effort	Strategic Learning

Example of Coding				1st level of code	2nd level of code (Indicators of Engagement)
Focus Group 1 New Zealand	Focus Group 2 New Zealand	Focus Group 3 New Zealand	Focus Group 4 Malaysia		
"...always excited to get a new program.. really fun" (MH, FG1) "...thoroughly enjoy programming.. thoroughly hate doing essays" (CM, FG1)		"Not frustration but definitely excitement when I finish [my programming assessment]." (RB, FG3) "I actually enjoy doing this subject and I want to like start on it [new programming assessment]" (JR, FG3)		• Excitement • Joy	Enjoyment
"..... I enjoy seeing it grow from a blank, empty file to presenting this thing on the screen and having it do my bidding so to speak. It's just a very satisfying feeling..." (CM, FG1)	"..You're getting somewhere while doing it [programming]. Sometimes you are writing 5 words [... other course] and you think do I agree with it. Do I think its right?" (DI, FG2) "..Just kind of smile thinking you've done it. It's finished." (DI, FG2)	"I think it's easier for me. Once the code is working, it's like you achieve something." (AL, FG3)		Programming is: • Rewarding all the way • Able to see immediate output • Feel a sense of achievement • Triumphant when problem solved	Gratification

Example of Coding				1st level of code	2nd level of code (Indicators of Engagement)
Focus Group 1 New Zealand	Focus Group 2 New Zealand	Focus Group 3 New Zealand	Focus Group 4 Malaysia		
"I like programming a lot, but that's not my life" (YL, FG1)			"I am interested because the code itself makes me very curious about what the outcome would be." (FA, FG4)	Programming is: • Cool • My vocation • My life • Stirs my curiosity • Makes me wonder about the logic	Interest

Appendix C: Survey Questionnaire (Phase 1)

Page 1



TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

Thank you for agreeing to participate in this survey. This survey should take no more than 10 minutes.

The goal of this research is to examine your personal attitudes in learning programming and how your personal attitudes might affect your performance in your programming course. Your responses to this research might help programming course instructors design better introductory programming courses in order to improve the learning experiences of programming students like you in the near future.

The Human Ethics Committee of the School of Information Management has approved this research project. Your participation in this project is voluntary and your responses to the information in this research project will be kept confidential. All data collected from you will be destroyed within 5 years after the completion of this research project.

This questionnaire is the first out of two questionnaires that ask you about your personal attitudes towards learning programming. Towards the end of your introductory programming course, I will be in contact with you again to ask you more about your personal attitudes towards learning programming.

This questionnaire has two sections. The first section asks you a series of questions about your personal attitudes when learning programming in the introductory programming course that you are enrolled in presently. The second section asks you some questions about yourself.

Please click on the forward arrow to take the survey.

0% 100%

<< >>

TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI VICTORIA UNIVERSITY OF WELLINGTON

Terminology

Assessment *– In this questionnaire, the term assessment refers to the work that was graded during your programming course. These may include assessment of your progress through a series of assignments, practical exercises, laboratory (lab) or workshop exercises, tutorial exercises, and your programming test or exam.*

Section 1: Personal attitudes towards learning programming

The following statements ask you about your level of confidence in tackling new tasks and learning programming. As you are now at the start of your introductory programming course, please rate **how confident are you now**, by using a scale of 1 (not confident at all) to 5 (very confident) in succeeding in these tasks **by the end** of your programming course.

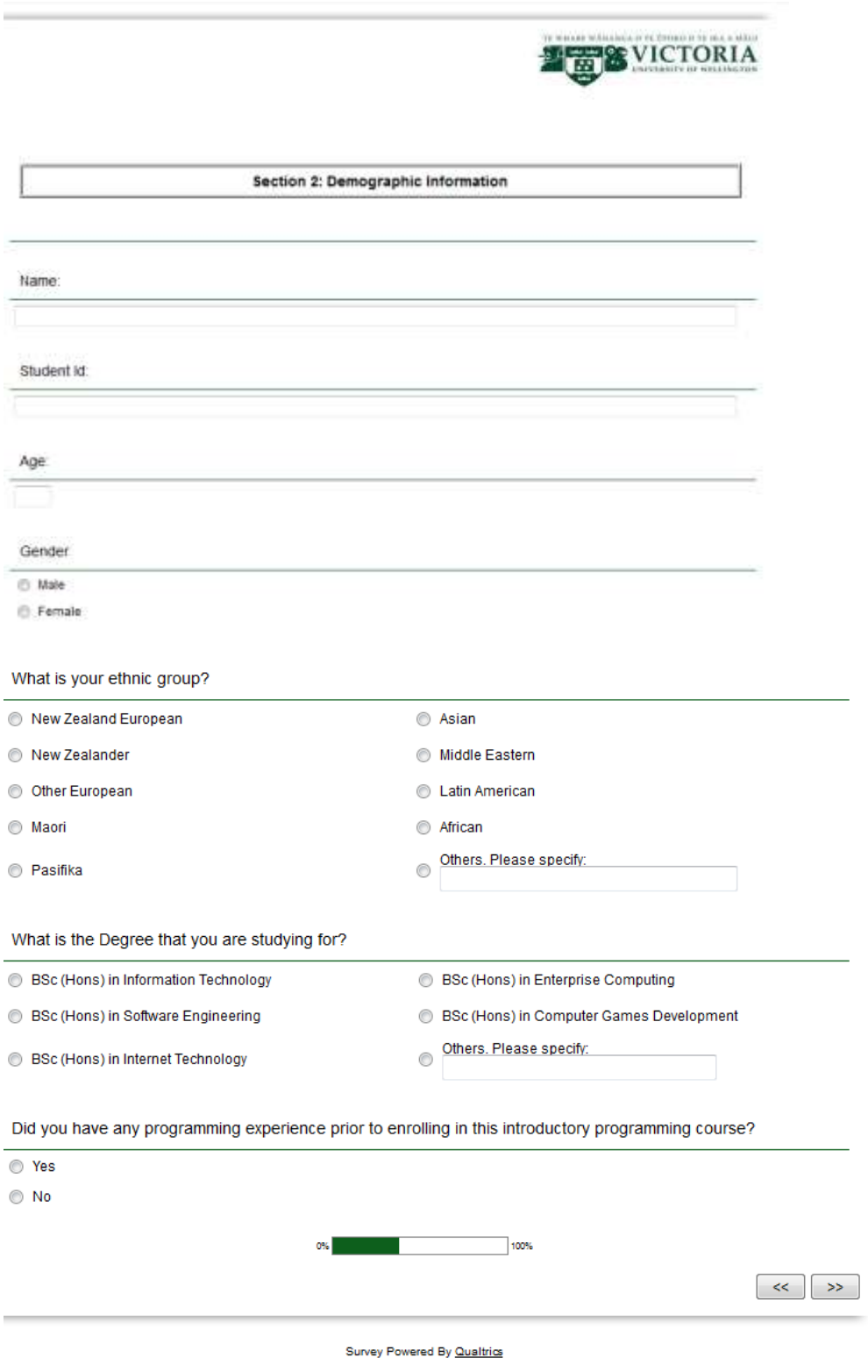
By the end of my programming course, I am confident that...

	Not confident at all 1....2....3....4....5 Very confident				
	1	2	3	4	5
I can complete a new task that is assigned to me if I keep trying.	○	○	○	○	○
I can stick to completing a task even though it may be unpleasant.	○	○	○	○	○
when I learn something new, I will not give up easily if I am not successful initially.	○	○	○	○	○
when unexpected problems occur, I can handle them well.	○	○	○	○	○
I can achieve the goals that I set for myself in this programming course.	○	○	○	○	○

Page 3 contd.

I have the capability to learn the contents of this programming course.	○	○	○	○	○
I can complete my programming assessment if I had a lot of time.	○	○	○	○	○
I can complete my programming assessment once someone else helps me get started.	○	○	○	○	○
I can complete my programming assessment if I can call someone for help if I get stuck.	○	○	○	○	○
I can complete my programming assessment if I refer to resources such as the built-in help, programming reference manuals, or online programming forums.	○	○	○	○	○
I can complete my programming assessment if someone showed me how to solve the problem first.	○	○	○	○	○
I can find ways of overcoming the problem if I get stuck at a point while working on my programming assessment.	○	○	○	○	○
I can correct (debug) all the errors in a program that I have written, and make it work.	○	○	○	○	○
I can apply the right programming concepts to solve a problem given to me.	○	○	○	○	○
I can understand and apply the fundamental object-oriented programming concepts used in my programming course.	○	○	○	○	○
I can make use of the pre-written library in the programming integrated development environment (IDE).	○	○	○	○	○
I can write a program to solve any given problem as long as the specifications are clear.	○	○	○	○	○
I can mentally trace through the execution of a complex program given to me.	○	○	○	○	○
I can find a way to concentrate on my programming assessment, even when there are many distractions around me.	○	○	○	○	○
I can find ways of motivating myself to program, even if the problem area was of no interest to me.	○	○	○	○	○
I can write programming code that run without errors (no syntax errors).	○	○	○	○	○
I can write programming code that is logical.	○	○	○	○	○
I can write a small program for a small problem that is familiar to me.	○	○	○	○	○
I can understand the language structure and the usage of the reserved words /keywords in the programming language.	○	○	○	○	○
I can write a reasonably sized program that can solve a problem that is only vaguely familiar to me.	○	○	○	○	○

<< >>



VICTORIA

Section 2: Demographic Information

Name:

Student Id:

Age:

Gender

- Male
- Female

What is your ethnic group?

- New Zealand European
- New Zealander
- Other European
- Maori
- Pasifika
- Asian
- Middle Eastern
- Latin American
- African
- Others. Please specify:

What is the Degree that you are studying for?

- BSc (Hons) in Information Technology
- BSc (Hons) in Software Engineering
- BSc (Hons) in Internet Technology
- BSc (Hons) in Enterprise Computing
- BSc (Hons) in Computer Games Development
- Others. Please specify:

Did you have any programming experience prior to enrolling in this introductory programming course?

- Yes
- No

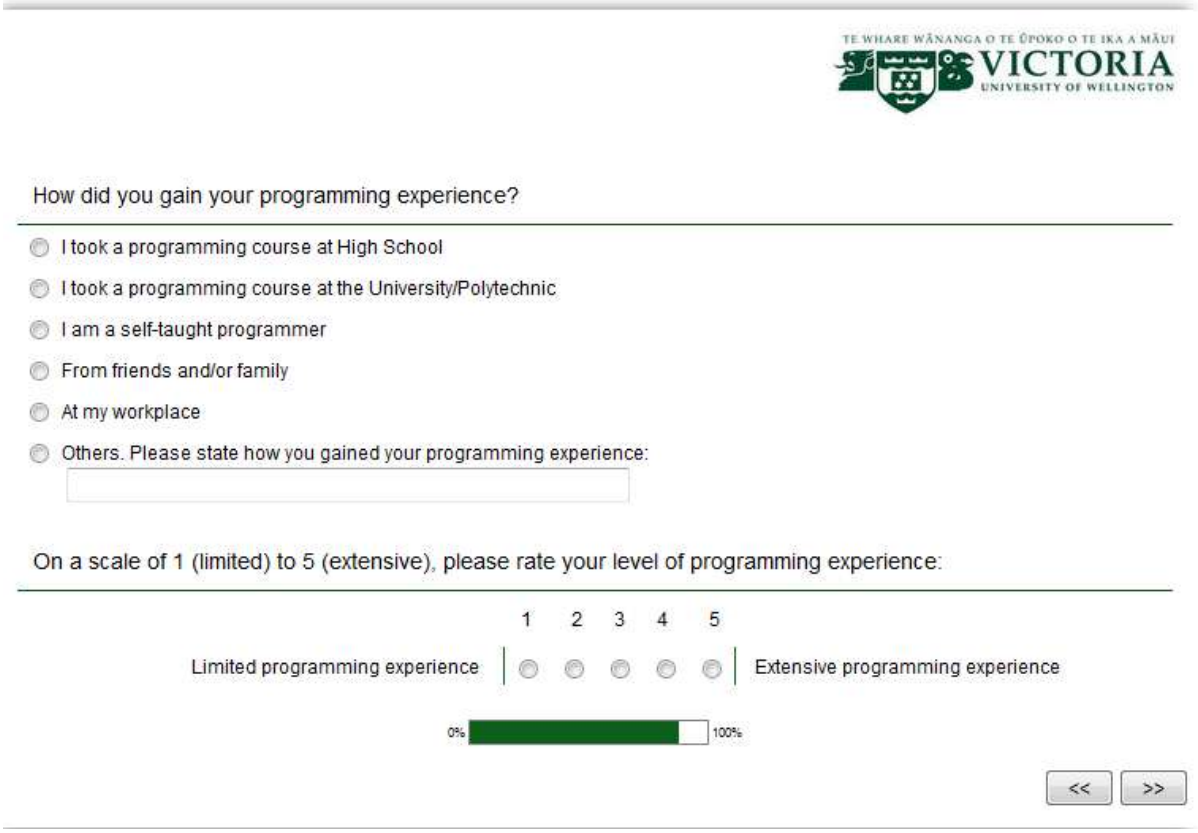
0% 100%

<< >>

Survey Powered By Qualtrics

Page 5

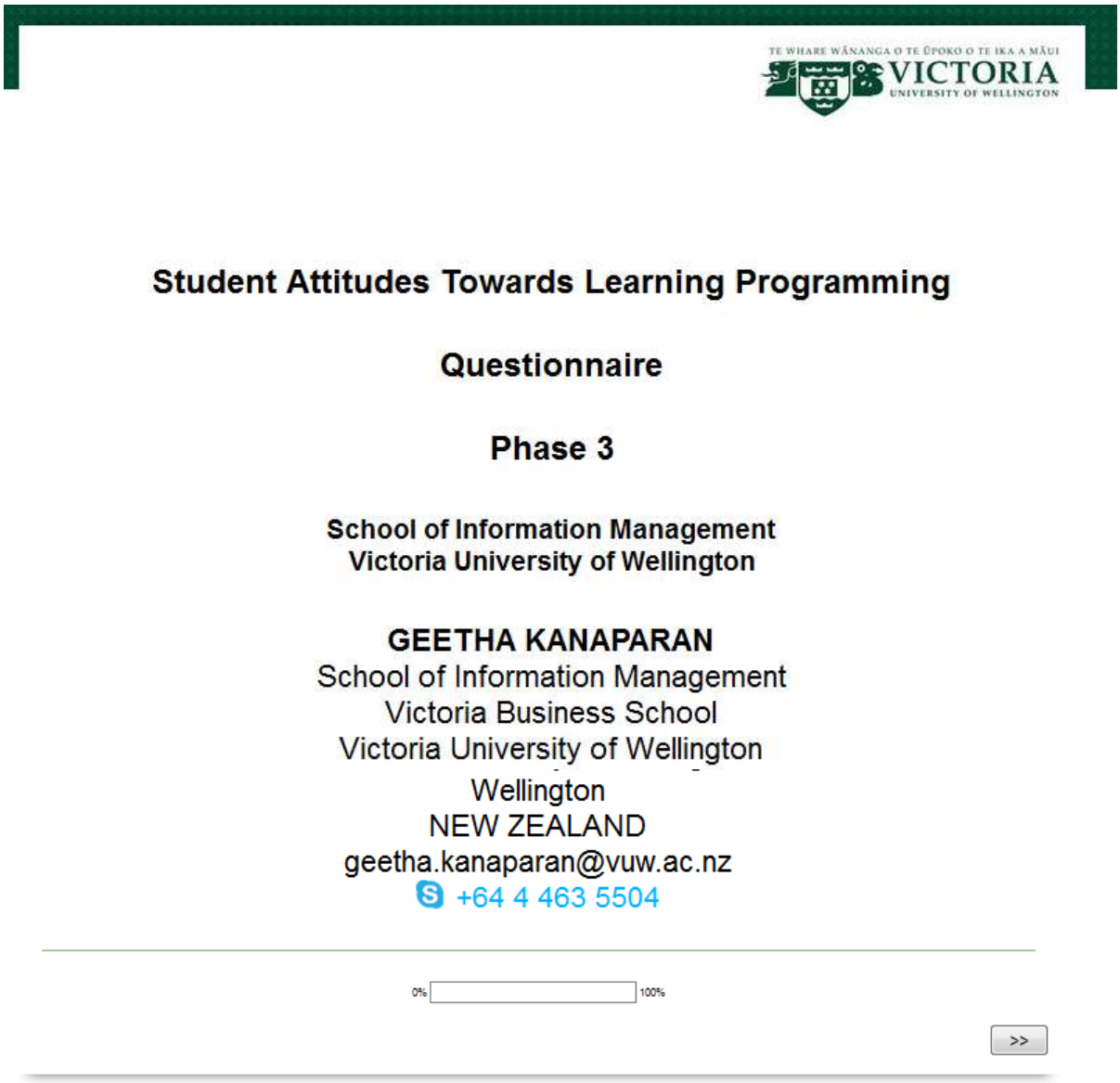
Note: This page is only displayed when the participant answers Yes to the question "Did you have any programming experience prior to enrolling in this introductory programming course?" in Page 4



Page 6

Appendix D: Survey Questionnaire (Phase 3)

Page 1



TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI VICTORIA UNIVERSITY OF WELLINGTON

Introduction

Thank you again for agreeing to participate in this survey. At the beginning of your introductory programming course, you participated in the first questionnaire that asked you about your personal attitudes towards learning programming. Now that you have completed your introductory programming course, this questionnaire asks you again about your personal attitudes towards learning programming.

The Human Ethics Committee of the School of Information Management has approved this research project. Your participation is voluntary and your responses to the information in this research project will be kept confidential. All data collected from you will be destroyed within 5 years after the completion of this research project. For further information, please see the Participant Information Sheet that was attached in the e-mail that contained the link to this questionnaire.

This questionnaire will take about 15 minutes to complete and has three sections. The first section asks you questions about your personal attitudes when learning programming in the introductory programming course that you are enrolled in. The second section asks you questions about how you engaged in your introductory programming course. The final section asks you questions about your performance in this course.

Please enter your Name :

Please enter your Student Id.:

Terminology

Assessment *– In this questionnaire, the term assessment refers to the work that was graded during your introductory programming course. These may include assessment of your progress through a series of assignments, practical exercises, laboratory (lab) or workshop exercises, tutorial exercises, and your programming test or exam.*

Practical(s) *– In this questionnaire, the term practical(s) refers to your workshop, tutorials, laboratory (lab), or practical sessions. Typically, these practical sessions require you to apply the concepts you learned during your lectures by writing code to solve programming problems. These practical sessions may or may not be formally assessed.*

Class *– Unless explicitly stated, the term class refers to your lecture and practical sessions ie. the formal contact hours specified in your course outline/syllabus.*

0% 100%

<< >>

TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI VICTORIA UNIVERSITY OF WELLINGTON

Terminology

Assessment *– In this questionnaire, the term assessment refers to the work that was graded during your introductory programming course. These may include assessment of your progress through a series of assignments, practical exercises, laboratory (lab) or workshop exercises, tutorial exercises, and your programming test or exam.*

Practical(s) *– In this questionnaire, the term practical(s) refers to your workshop, tutorials, laboratory (lab), or practical sessions. Typically, these practical sessions require you to apply the concepts you learned during your lectures by writing code to solve programming problems. These practical sessions may or may not be formally assessed.*

Class *– Unless explicitly stated, the term class refers to your lecture and practical sessions ie. the formal contact hours specified in your course outline/syllabus.*

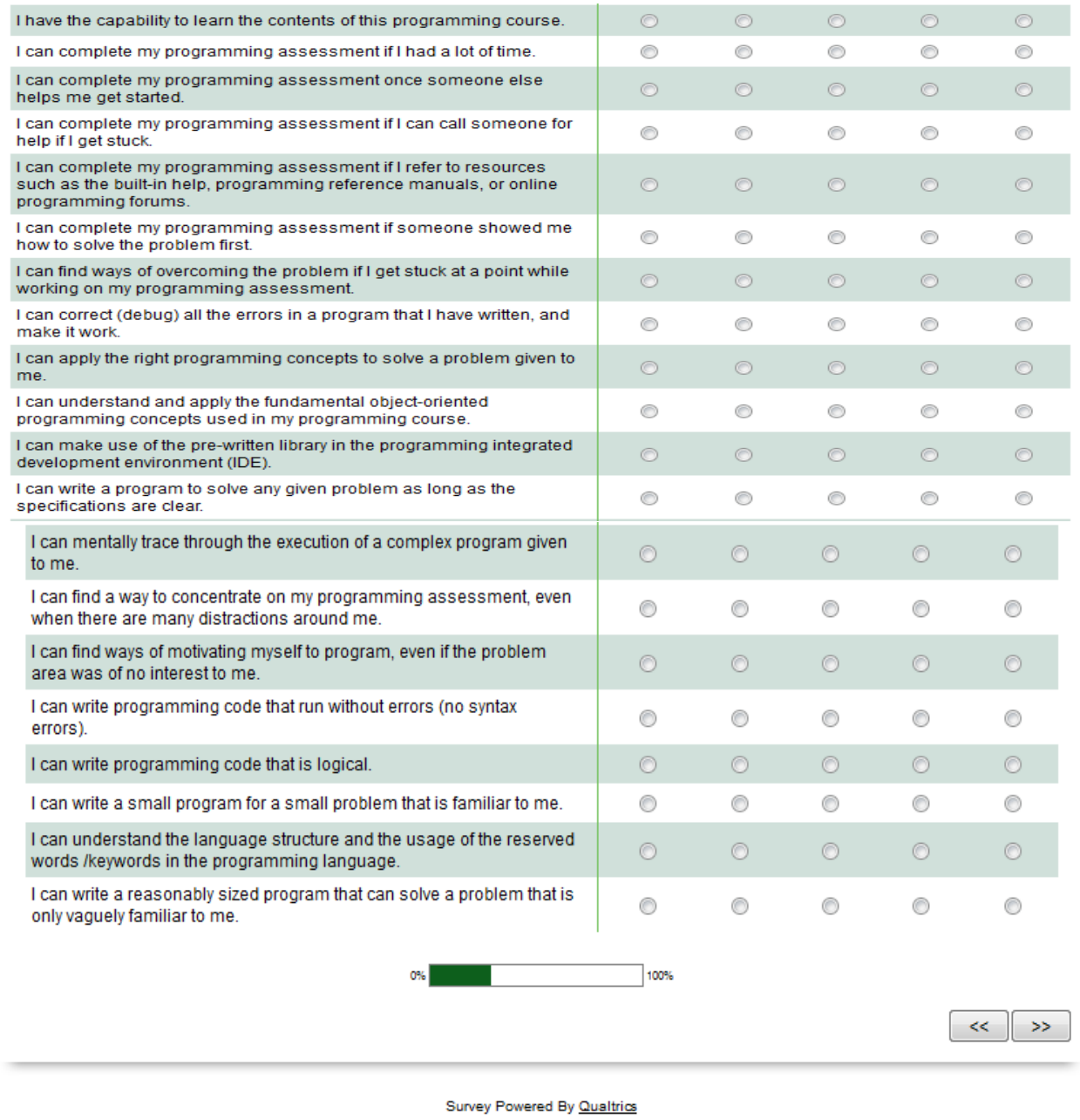
Section 1: Personal Attitudes towards Learning Programming

At the beginning of your introductory programming course, you may have answered questions that were similar to the ones in this section. Now that you are at the end of your introductory programming course, please reflect on how you have progressed through your introductory programming course and answer the following statements about your current level of confidence in tackling new tasks and learning programming. By using a scale of 1 (not confident at all) to 5 (very confident), please indicate to what extent are you confident in succeeding in these tasks.

I am confident that...

	Not confident at all 1....2....3....4....5 Very confident				
	1	2	3	4	5
I can complete a new task that is assigned to me if I keep trying.	○	○	○	○	○
I can stick to completing a task even though it may be unpleasant.	○	○	○	○	○
when I learn something new, I will not give up easily if I am not successful initially.	○	○	○	○	○
when unexpected problems occur, I can handle them well.	○	○	○	○	○
I can achieve the goals that I set for myself in this programming course.	○	○	○	○	○

Page 3 contd.



I have the capability to learn the contents of this programming course.					
I can complete my programming assessment if I had a lot of time.					
I can complete my programming assessment once someone else helps me get started.					
I can complete my programming assessment if I can call someone for help if I get stuck.					
I can complete my programming assessment if I refer to resources such as the built-in help, programming reference manuals, or online programming forums.					
I can complete my programming assessment if someone showed me how to solve the problem first.					
I can find ways of overcoming the problem if I get stuck at a point while working on my programming assessment.					
I can correct (debug) all the errors in a program that I have written, and make it work.					
I can apply the right programming concepts to solve a problem given to me.					
I can understand and apply the fundamental object-oriented programming concepts used in my programming course.					
I can make use of the pre-written library in the programming integrated development environment (IDE).					
I can write a program to solve any given problem as long as the specifications are clear.					
I can mentally trace through the execution of a complex program given to me.					
I can find a way to concentrate on my programming assessment, even when there are many distractions around me.					
I can find ways of motivating myself to program, even if the problem area was of no interest to me.					
I can write programming code that run without errors (no syntax errors).					
I can write programming code that is logical.					
I can write a small program for a small problem that is familiar to me.					
I can understand the language structure and the usage of the reserved words /keywords in the programming language.					
I can write a reasonably sized program that can solve a problem that is only vaguely familiar to me.					

TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI VICTORIA UNIVERSITY OF WELLINGTON

Terminology

Assessment *– In this questionnaire, the term assessment refers to the work that was graded during your introductory programming course. These may include assessment of your progress through a series of assignments, practical exercises, laboratory (lab) or workshop exercises, tutorial exercises, and your programming test or exam.*

Practical(s) *– In this questionnaire, the term practical(s) refers to your workshop, tutorials, laboratory (lab), or practical sessions. Typically, these practical sessions require you to apply the concepts you learned during your lectures by writing code to solve programming problems. These practical sessions may or may not be formally assessed.*

Class *– Unless explicitly stated, the term class refers to your lecture and practical sessions ie. the formal contact hours specified in your course outline/syllabus.*

Section 2: Engagement

This section asks you about your behaviour, thinking processes, and your feelings during your introductory programming course. When answering the questions in this section, please reflect on how you progressed through your introductory programming course. Please indicate to what extent you agree with these statements by using a scale of 1 (strongly disagree) to 5 (strongly agree). Where the statement does not apply to your course, please select N/A (not applicable).

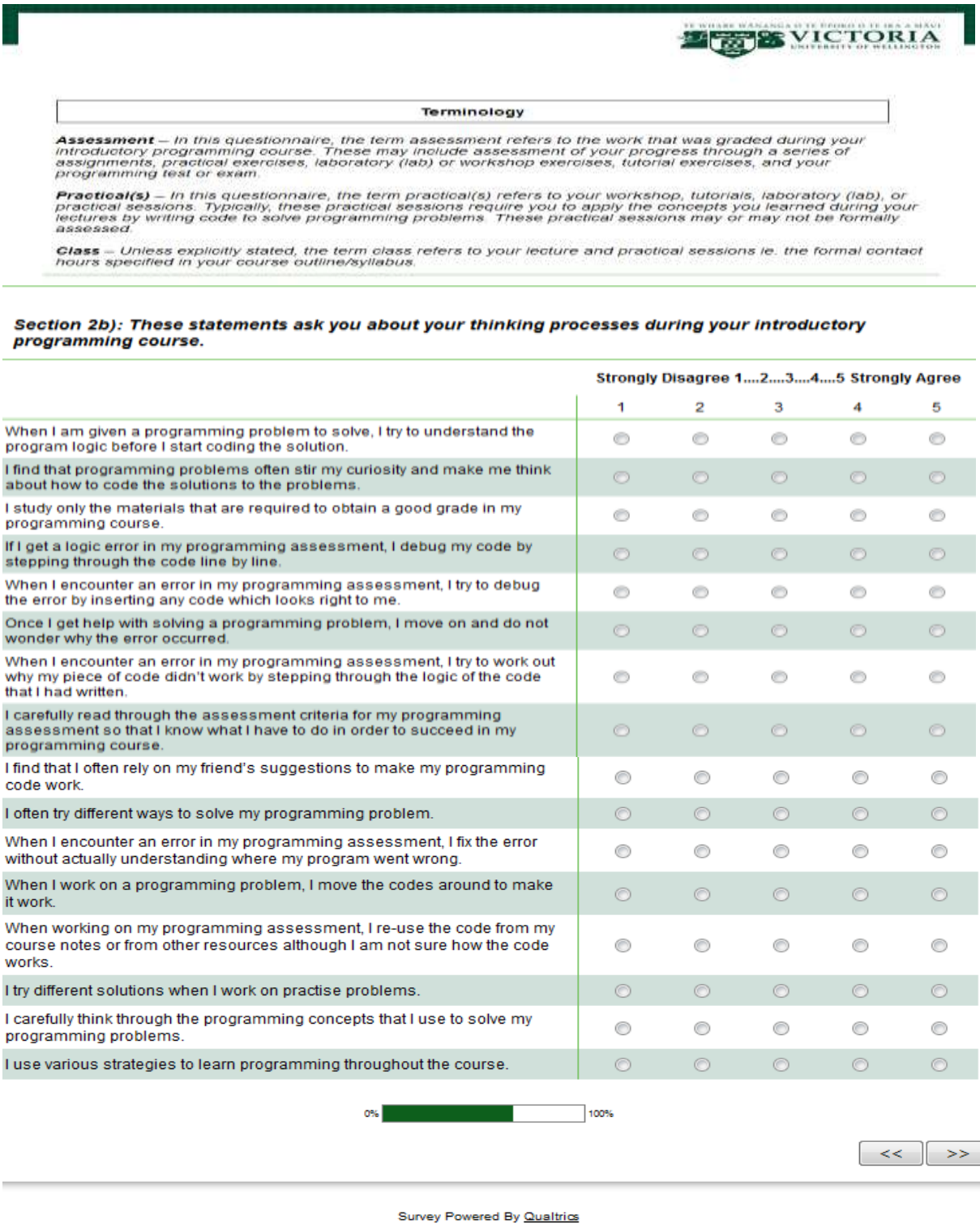
Section 2a): These statements ask you about your behaviour during your introductory programming course.

	Strongly Disagree 1....2....3....4....5 Strongly Agree				
	1	2	3	4	5
I try to stay on top of my programming assessments by starting and completing them early.	○	○	○	○	○
I expect to be given a hint to the solution when I ask for help to debug my programming errors.	○	○	○	○	○
When I don't understand a programming concept, I take time to review my lecture notes and online resources that would help me understand the concept.	○	○	○	○	○
I often practise writing programs in order to understand a programming concept.	○	○	○	○	○
I get help whenever I can to get through my programming assessments.	○	○	○	○	○
I worked all the way through the course to complete all of my programming assessments.	○	○	○	○	○
Asking for help would be one of the first things I would do if I had errors in my programming assessment.	○	○	○	○	○
When I don't understand a programming problem, I keep practicing until I understand the problem.	○	○	○	○	○
When my program doesn't work, I am determined to fix the errors no matter how long it takes.	○	○	○	○	○
I can spend hours debugging an error in my programming assessment without feeling like giving up.	○	○	○	○	○
I review my lecture notes and refer to other resources again and again in order to try and understand a difficult programming concept of problem.	○	○	○	○	○
I organise my time effectively in order to maximize the effort spent on my programming assessment.	○	○	○	○	○
I was always involved in learning programming throughout the course.	○	○	○	○	○
I was constantly working on my programming assessments during the course.	○	○	○	○	○

0% 100%

<< >>

Survey Powered By Qualtrics



VICTORIA
UNIVERSITY OF WELLINGTON

Terminology

Assessment *– In this questionnaire, the term assessment refers to the work that was graded during your introductory programming course. These may include assessment of your progress through a series of assignments, practical exercises, laboratory (lab) or workshop exercises, tutorial exercises, and your programming test or exam.*

Practical(s) *– In this questionnaire, the term practical(s) refers to your workshop, tutorials, laboratory (lab), or practical sessions. Typically, these practical sessions require you to apply the concepts you learned during your lectures by writing code to solve programming problems. These practical sessions may or may not be formally assessed.*

Class *– Unless explicitly stated, the term class refers to your lecture and practical sessions ie. the formal contact hours specified in your course outline/syllabus.*

Section 2b): These statements ask you about your thinking processes during your introductory programming course.

	Strongly Disagree 1....2....3....4....5 Strongly Agree				
	1	2	3	4	5
When I am given a programming problem to solve, I try to understand the program logic before I start coding the solution.					
I find that programming problems often stir my curiosity and make me think about how to code the solutions to the problems.					
I study only the materials that are required to obtain a good grade in my programming course.					
If I get a logic error in my programming assessment, I debug my code by stepping through the code line by line.					
When I encounter an error in my programming assessment, I try to debug the error by inserting any code which looks right to me.					
Once I get help with solving a programming problem, I move on and do not wonder why the error occurred.					
When I encounter an error in my programming assessment, I try to work out why my piece of code didn't work by stepping through the logic of the code that I had written.					
I carefully read through the assessment criteria for my programming assessment so that I know what I have to do in order to succeed in my programming course.					
I find that I often rely on my friend's suggestions to make my programming code work.					
I often try different ways to solve my programming problem.					
When I encounter an error in my programming assessment, I fix the error without actually understanding where my program went wrong.					
When I work on a programming problem, I move the codes around to make it work.					
When working on my programming assessment, I re-use the code from my course notes or from other resources although I am not sure how the code works.					
I try different solutions when I work on practise problems.					
I carefully think through the programming concepts that I use to solve my programming problems.					
I use various strategies to learn programming throughout the course.					

0% 100%

<< >>

Survey Powered By Qualtrics

Terminology

Assessment *– In this questionnaire, the term assessment refers to the work that was graded during your introductory programming course. These may include assessment of your progress through a series of assignments, practical exercises, laboratory (lab) or workshop exercises, tutorial exercises, and your programming test or exam.*

Practical(s) *– In this questionnaire, the term practical(s) refers to your workshop, tutorials, laboratory (lab), or practical sessions. Typically, these practical sessions require you to apply the concepts you learned during your lectures by writing code to solve programming problems. These practical sessions may or may not be formally assessed.*

Class *– Unless explicitly stated, the term class refers to your lecture and practical sessions ie. the formal contact hours specified in your course outline/syllabus.*

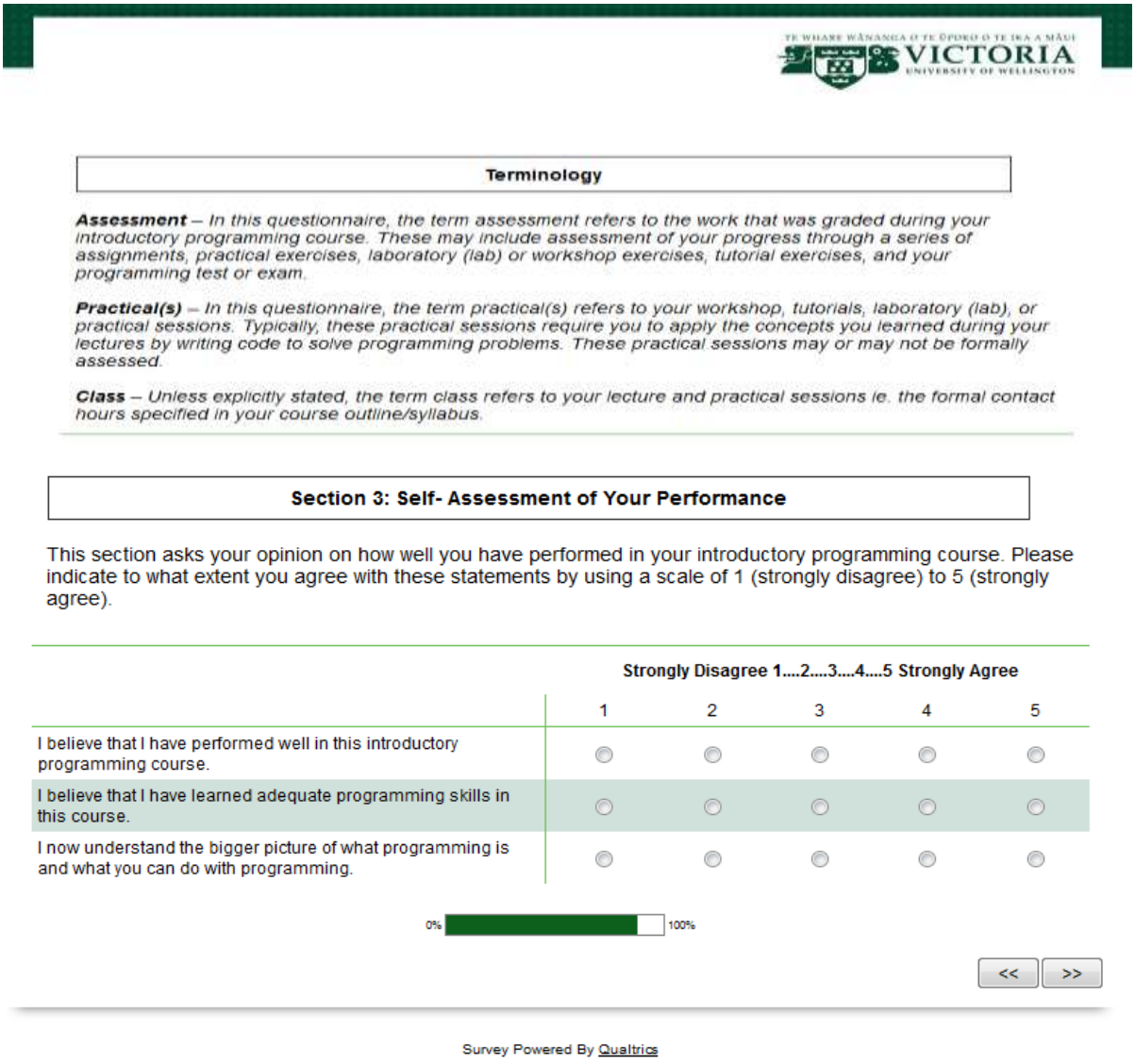
Section 2c): These statements ask you about your feelings during your introductory programming course.

	Strongly Disagree 1....2....3....4....5 Strongly Agree				
	1	2	3	4	5
My programming assessments take priority over the assessments from other courses.	○	○	○	○	○
My programming course is stimulating compared to the other courses that I am enrolled in.	○	○	○	○	○
Once I complete my programming assessment, I feel pleased that I have successfully completed a challenging piece of assessment.	○	○	○	○	○
I feel a deep sense of satisfaction when I finally get my program to work.	○	○	○	○	○
I am passionate about programming.	○	○	○	○	○
The challenge of coding solutions to programming problems appeals to me.	○	○	○	○	○
I feel relieved when I complete my programming assessment.	○	○	○	○	○
My programming course suits me better than the other courses that I am currently enrolled in.	○	○	○	○	○
I like writing programs.	○	○	○	○	○
I feel a strong sense of achievement when I complete my programming assessments.	○	○	○	○	○
Programming stirs up positive emotions in me.	○	○	○	○	○

<< >>

Survey Powered By Qualtrics

Page 7



Terminology

Assessment *– In this questionnaire, the term assessment refers to the work that was graded during your introductory programming course. These may include assessment of your progress through a series of assignments, practical exercises, laboratory (lab) or workshop exercises, tutorial exercises, and your programming test or exam.*

Practical(s) *– In this questionnaire, the term practical(s) refers to your workshop, tutorials, laboratory (lab), or practical sessions. Typically, these practical sessions require you to apply the concepts you learned during your lectures by writing code to solve programming problems. These practical sessions may or may not be formally assessed.*

Class *– Unless explicitly stated, the term class refers to your lecture and practical sessions ie. the formal contact hours specified in your course outline/syllabus.*

Section 3: Self- Assessment of Your Performance

This section asks your opinion on how well you have performed in your introductory programming course. Please indicate to what extent you agree with these statements by using a scale of 1 (strongly disagree) to 5 (strongly agree).

	Strongly Disagree 1....2....3....4....5 Strongly Agree				
	1	2	3	4	5
I believe that I have performed well in this introductory programming course.	○	○	○	○	○
I believe that I have learned adequate programming skills in this course.	○	○	○	○	○
I now understand the bigger picture of what programming is and what you can do with programming.	○	○	○	○	○

0% 100%

<< >>

Survey Powered By Qualtrics

Page 8

Thank you for your participation in this research.

If you have any questions on this survey, please contact:

GEETHA KANAPARAN
School of Information Management
Victoria Business School
Victoria University of Wellington
Wellington
NEW ZEALAND
Geetha.Kanaparan@vuw.ac.nz
+64 4 463 5504

0% 100%

Survey Powered By Qualtrics

Appendix E: Participant Consent Forms

Phase 1 & Phase 3 (Survey Questionnaire)

SCHOOL OF INFORMATION MANAGEMENT
TE KURA TIAKI, WHAKAWHITI KŌRERO
LEVEL 5, RUTHERFORD HOUSE, PIPITEA CAMPUS, 23 LAMBTON QUAY, WELLINGTON
PO Box 600, Wellington 6140, New Zealand
Phone + 64-4-463 5103 Fax +64-4-463 5446 Email sim@vuw.ac.nz Website www.victoria.ac.nz/sim

Participant Consent Form

Research Project Title: Self-Efficacy and Engagement as Predictors of Student Programming Performance

Researcher: Geetha Kanaparan, School of Information Management, Victoria Business School, Victoria University of Wellington

I have been given and have understood an explanation of this research project. I have had an opportunity to ask questions and have them answered to my satisfaction.

I understand that my participation is entirely voluntary, and that my non-participation will have no bearing on the final grade of my course.

I understand that I may withdraw myself (or any information I have provided) from this project, without having to give reasons, by e-mailing Geetha.Kanaparan@vuw.ac.nz by the [1 week after data is collected at the end of the programming course].

I understand that any information I provide will be kept confidential to the researcher and their supervisor, the published results will not use my name, and that no opinions will be attributed to me in any way that will identify me with the higher educational institution that I am enrolled in.

I understand that the data I provide will not be used for any other purpose or released to others. However, my data may be provided in an anonymised and aggregated form if it is requested by my Institution.

I understand that the data will be erased within 5 years after the conclusion of the research project.

Please indicate (by ticking the boxes below) which of the following apply:

- ☐ I would like to receive a summary of the results of this research when it is completed.
  Please provide your e-mail address: _____
- ☐ I agree to participate in the survey questionnaire at the beginning and at the end of my programming course
  Please provide your e-mail address: _____
- ☐ I agree to the researcher accessing my school results, my programming course grades, and my grades from other courses that I took this term from the Faculty's student database
- ☐ I would like to be entered into the lucky draw for a cash voucher of NZD$50

Signed:
Name of participant:
Student Id.:
Date:

Phase 2 (Focus Groups)

SCHOOL OF INFORMATION MANAGEMENT
TE KURA TIAKI, WHAKAWHITI KŌRERO
LEVEL 5, RUTHERFORD HOUSE, PIPITEA CAMPUS, 23 LAMBTON QUAY, WELLINGTON
PO Box 600, Wellington 6140, New Zealand
Phone + 64-4-463 5103 Fax +64-4-463 5446 Email sim@vuw.ac.nz Website www.victoria.ac.nz/sim

Participant Consent Form
for
Focus Groups

Research Project Title: Self-Efficacy and Engagement as Predictors of Student Programming Performance

Researcher: Geetha Kanaparan, School of Information Management, Victoria Business School, Victoria University of Wellington

I have been given and have understood an explanation of this research project. I have had an opportunity to ask questions and have them answered to my satisfaction.

I understand that my participation is entirely voluntary, and that my non-participation will have no bearing on the final grade of my course.

I understand that it is not possible to extract my comment from the focus group discussion, and that I will not be able withdraw myself (or any information I have provided) from the focus group recording.

I understand that the focus group discussions are confidential, and that I will not disclose anything that was said during the focus groups sessions to anyone outside of the focus group.

I understand that any information I provide will be kept confidential to the researcher and their supervisor, the published results will not use my name, and that no opinions will be attributed to me in any way that will identify me with the higher educational institution that I am enrolled in.

I understand that the data I provide will not be used for any other purpose or released to others.

I understand that the recording and transcripts of the interviews will be erased within 5 years after the conclusion of the research project.

Signed:

Name of participant:

Date:

Appendix F: Card Sorting Instructions

TE WHARE WĀNANGA O TE ĀPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

Activity: Card Sorting (Round 1)

Research Project Title: **Self-Efficacy and Engagement as Predictors of Student Programming Performance**

Researcher: **Geetha Kanaparan, School of Information Management, Victoria Business School, Victoria University of Wellington**

Dear Sir/Madam,

Thank you for taking the time to participate in this card sorting activity. The purpose of this card sorting activity is to help me design better scale items for the survey instrument of my research project.

The objective of my research project is to examine the personal attitudes of students when they learn programming and how their personal attitudes might affect their performance in their introductory programming course. I hope that the outcome of this study will help programming course instructors understand how the personal attitudes of students affect their performance in programming.

This card sorting activity should take about 45 minutes. With this cover letter, you will have a set of individual statements given to you in no particular order. These statements explain the activities that students undertake, or experiences of students in their introductory programming course. Your task is to group these statements into sets of meaningful activities or experiences that a student might undertake when learning programming. Once you have grouped these statements, write what kind of task or experience that you feel each group of statements represent. If there are ambiguous statements, or if you are unable to group a statement, please set the statement aside as a separate set of statements and write your comment at the end of your card sorting activity sheet.

As the survey instrument is still being developed and refined, I would appreciate it if you could keep your participation in this card sorting activity confidential. Thank you again for your participation. If you have any comments or suggestions with regards to the card sorting activity, please do contact me personally or write your comment at the end of your card sorting activity.

Geetha Kanaparan
PhD Student
School of Information Management
Victoria Business School
Victoria University of Wellington
Wellington
NEW ZEALAND
Geetha.Kanaparan@vuw.ac.nz
+64 4 463 5504

TE WHARE WĀNANGA O TE ĀPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

Activity: Card Sorting (Round 2)

Research Project Title: **Self-Efficacy and Engagement as Predictors of Student Programming Performance**

Researcher: **Geetha Kanaparan, School of Information Management, Victoria Business School, Victoria University of Wellington**

Dear Sir/Madam,

Thank you for taking the time to participate in this card sorting activity. The purpose of this card sorting activity is to help me design better scale items for the survey instrument of my research project.

The objective of my research project is to examine the personal attitudes of students when they learn programming and how their personal attitudes might affect their performance in their introductory programming course. I hope that the outcome of this study will help programming course instructors understand how the personal attitudes of students affect their performance in programming.

This card sorting activity should take about 30 minutes. With this cover letter, you will have two sets of cards. In the first set of cards given to you, there will be statement on each card that describes an activity that a student undertakes or the experience of a student in an introductory programming course. In the second set, each card contains a name and definition of an activity, or an experience which may be used to explain each statement. Your task is to match the name and definition of the activity or experience, to the statements. If there are ambiguous statements, or if you are unable to group a statement, please set the statement aside as a separate set of statements and write your comment at the end of your card sorting activity sheet.

As the survey instrument is still being developed and refined, I would appreciate it if you could keep your participation in this card sorting activity confidential. Thank you again for your participation. If you have any comments or suggestions, please do contact me personally or write your comment at the end of your card sorting activity sheet.

Geetha Kanaparan
PhD Student
School of Information Management
Victoria Business School
Victoria University of Wellington
Wellington
NEW ZEALAND
Geetha.Kanaparan@vuw.ac.nz
+64 4 463 5504

Appendix G: Multiple Mediation

The objective of the test for multiple mediation was two-fold. The first objective was to test whether each of the engagement constructs mediated the relationship between *post-programming self-efficacy* and *programming grade*, and between *post-programming self-efficacy* and *self-assessment*. The second objective was to test whether the engagement constructs cumulatively mediated the relationships between *post-programming self-efficacy* and *programming grade*, and between *post-programming self-efficacy* and *self-assessment*.

There are four methods for mediation analysis. They are the *causal steps strategy* (Baron & Kenny, 1986), *Sobel test* (Sobel, 1982), *distribution of the product* (MacKinnon et al., 2002), and *bootstrapping* (Preacher & Hayes, 2008). The bootstrapping method was used to test for multiple mediation as this method does not assume that the data is normally distributed (Preacher & Hayes, 2008). According to MacKinnon (2008), the Baron and Kenny (1986) test for single mediation may also be used to test multiple mediation, but with some constraints. MacKinnon (2008) recommends four tests for multiple mediation and is presented in Table AG.1.

Table AG.1: Tests for multiple mediation

Test	Description
Test 1	Test if the exogenous variable *post-programming self-efficacy* affects the endogenous variables *programming grade* and *self-assessment*. There should be a significant relationship between them. If this effect is not significant, then the analysis for mediation should not proceed.
Test 2	Test if the exogenous variable *post-programming self-efficacy* affects the mediators - *behavioural engagement*, *cognitive engagement*, and *emotional engagement*. There should be a significant relationship between them.
Test 3	Test if the mediators *behavioural engagement*, *cognitive engagement*, and *emotional engagement* affects the endogenous variables *programming grade* and/or *self-assessment* when the exogenous variable *post-programming self-efficacy* is controlled. There should be a significant relationship between the mediators and the endogenous variables.
Test 4	The direct effect should not be significant.

The multiple mediation test was performed in SPSS (Version 20.0) using the Preacher and Hayes (2008) Multiple Mediation Test using Bootstrapping. The multiple mediation tests were carried out separately for *programming grade*, and *self-assessment*. Table AG.2 presents the variables that were used to test for multiple mediation. The higher-order formative engagement constructs were used as the mediating variables.

Table AG.2: Variables used for multiple mediation

Type of construct	Name of Construct
Independent Variable	Post-programming self-efficacy
Mediating Variables	Behavioural Engagement Cognitive Engagement Emotional Engagement
Dependent Variable	Programming Grade Self-Assessment

Analysis 1: Mediation for Programming Grade

At a 95% confidence level, and at 5000 bootstrapping samples:

Test 1: Based on the output in Table AG.3, the total effect from *post-programming self-efficacy* to *programming grade* (without the mediating variables) was significant ($p = .000$). The result in Table AG.3 satisfies Test 1 – there is a significant relationship between the exogenous variable *post-programming self-efficacy* and the endogenous variable *programming grade*. Since Test 1 was significant, we then proceed to Step 2.

Table AG.3: Output for total effect from exogenous variable to the endogenous variable (without the mediating variable)

	Path Coefficient	Standard Error	t-value	p-value
Post-programming self-efficacy	.416	.044	9.488***	.000

Note: t-values > 1.65 (p<0.1); t-values > 1.96** (p<0.05); t-values > 2.57*** (p<0.01)*

Test 2: Based on the output in Table AG.4 (exogenous variable to mediating variables), the path from *post-programming self-efficacy* to *behavioural engagement* was significant ($p = .000$). The path from *post-programming self-efficacy* to *cognitive engagement* was significant ($p = .000$). The path from *post-programming self-efficacy* to *emotional engagement* was significant ($p = .000$). The result in Table AG.4 satisfied Test 2 (there is a significant relationship between the exogenous variable and the mediating variables).

Table AG.4: Output from Exogenous Variable to Mediating Variables

	Path Coefficient	Standard Error	t-value	p-value
Behavioural Engagement	.715	.034	21.228***	.000
Cognitive Engagement	.725	.033	21.823***	.000
Emotional Engagement	.654	.036	17.948***	.000

Note: t-values > 1.65 (p<0.1); t-values > 1.96** (p<0.05); t-values > 2.57*** (p<0.01)*

Test 3: Based on the output in Table AG.5 (mediating variables to endogenous variable while controlling for the exogenous variable), the path from *behavioural engagement* to *programming grade* was not significant ($p = .309$). The path from *cognitive engagement* to *programming*

grade was not significant ($p = .192$). The path from *emotional engagement* to *programming grade* was not significant ($p = .474$). The result in Table AG.5 did not satisfy Test 3. The relationship between the mediating engagement variables and the endogenous variable *programming grade* (while controlling for the exogenous variable *post-programming self-efficacy*) was not significant.

Table AG.5: Output for direct effects of mediators on endogenous variable

	Path Coefficient	Standard Error	t-value	p-value
Behavioural Engagement	.071	.070	1.018	.309
Cognitive Engagement	.093	.071	1.308	.192
Emotional Engagement	-.046	.064	-.716	.474

Note: t-values > 1.65 (p<0.1); t-values > 1.96** (p<0.05); t-values > 2.57*** (p<0.01)*

Test 4: Based on the output in Table AG.6, the direct effect from *post-programming self-efficacy* to *programming grade* (while controlling for the mediating variables) was significant ($p = 0.000$). The result in Table AG.6 did not satisfy Test 4. There should be a non-significant direct effect between the exogenous variable *post-programming self-efficacy* and the endogenous variable *programming grade*.

Table AG.6: Output for direct effect from exogenous variable to the endogenous variable (while controlling for the mediating variables)

	Path Coefficient	Standard Error	t-value	p-value
Post-programming self-efficacy	.327	.072	4.558***	.000

Note: t-values > 1.65 (p<0.1); t-values > 1.96** (p<0.05); t-values > 2.57*** (p<0.01)*

Next, the R^2 for *programming grade* was .180, which is the amount of variance in the endogenous variable (*programming grade*) accounted for by the exogenous variable (*post-programming self-efficacy*) and the mediating variables (*behavioural engagement*, *cognitive engagement*, and *emotional engagement*).

In the final step of determining if mediation exists, the confidence intervals were examined. Table AG.7 presents the lower and upper limits of the confidence intervals for the endogenous variable *programming grade*. If a zero (0) lies in between the confidence intervals, then no mediation exists. In Table AG.7, none of the indirect effect contained a 0 in between the confidence intervals. This results shows that the engagement constructs did not cumulatively mediate the relationship between *post-programming self-efficacy* and *programming grade*. Additionally, when the engagement constructs were analysed individually, none of the engagement constructs mediated the relationship between *post-programming self-efficacy* and *programming grade*.

Table AG.7: Confidence Intervals for endogenous variable programming grade

	Lower	Upper	Mediation?
Total	-.035	.208	No
Behavioural Engagement	-.043	.143	No
Cognitive Engagement	-.031	.170	No
Emotional Engagement	-.109	.051	No

Analysis 2: Mediation for Self-Assessment

At a 95% confidence level, and at 5000 bootstrapping samples:

Test 1: Based on the output in Table AG.8, the total effect from *post-programming self-efficacy* to *self-assessment* (without the mediating variables) is significant ($p = .000$). The result in Table AG.8 satisfied Test 1 – there is a significant relationship between the exogenous variable *post-programming self-efficacy* and the endogenous variable *self-assessment*. As Test 1 was significant, we then proceed to Step 2.

Table AG.8: Output for total effect from exogenous variable to the endogenous variable (without the mediating variable)

	Path Coefficient	Standard Error	t-value	p-value
Post-programming self-efficacy	.695	.034	20.826***	.000

Note: t-values > 1.65 (p<0.1); t-values > 1.96** (p<0.05); t-values > 2.57*** (p<0.01)*

Test 2: Based on the output in Table AG.9 (exogenous variable to mediating variables), the path from *post-programming self-efficacy* to *behavioural engagement* is significant ($p = .000$). The path from *post-programming self-efficacy* to *cognitive engagement* is significant ($p = .000$). The path from *post-programming self-efficacy* to *emotional engagement* is significant ($p = .000$). The result in Table AG.9 satisfied Test 2. There is a significant relationship between the exogenous variable and the mediating variables.

Table AG.9: Output from Exogenous Variable to Mediating Variables

	Path Coefficient	Standard Error	t-value	p-value
Behavioural Engagement	.715	.034	21.228***	.000
Cognitive Engagement	.725	.033	21.823***	.000
Emotional Engagement	.654	.036	17.948***	.000

Note: t-values > 1.65 (p<0.1); t-values > 1.96** (p<0.05); t-values > 2.57*** (p<0.01)*

Test 3: Based on the output in Table AG.10 (mediating variables to endogenous variable while controlling for the exogenous variables), the path from *behavioural engagement* to *self-assessment* was significant ($p = .023$). The path from *cognitive engagement* to *self-assessment* was not significant ($p = .281$), while the path from *emotional engagement* to *self-assessment* was significant ($p = .000$).

The result in Table AG.10 satisfied Test 3 for the relationship between *behavioural engagement* and *self-assessment* and the relationship between *emotional engagement* and *self-assessment*. There was a significant relationship between the mediating variable *emotional engagement* and the endogenous variable *self-assessment* (while controlling for the exogenous variable *post-programming self-efficacy*). This relationship was significant at a 1% probability of error. There was also a significant relationship between the mediating variable *behavioural engagement* and the endogenous variable *self-assessment* (while controlling for the exogenous variable *post-programming self-efficacy*). This relationship was significant at a 5% probability of error.

Table AG.10: Output for direct effects of mediators on endogenous variable

	Path Coefficient	Standard Error	t-value	p-value
Behavioural Engagement	.118	.052	2.280**	.023
Cognitive Engagement	.057	.053	1.081	.281
Emotional Engagement	.207	.047	4.372***	.000

Note: t-values > 1.65 (p<0.1); t-values > 1.96** (p<0.05); t-values > 2.57*** (p<0.01)*

Test 4: Based on the output in Table AG.11, the direct effect from *post-programming self-efficacy* to *self-assessment* (while controlling for the mediating variables) is significant ($p = 0.000$). The result in Table AG.11 did not satisfy Test 4. There should be a non-significant direct effect between the exogenous variable *post-programming self-efficacy* and the endogenous variable *self-assessment*.

Table AG.11: Output for direct effect from exogenous variable to the endogenous variable (while controlling for the mediating variables)

	Path Coefficient	Standard Error	t-value	p-value
Post-programming self-efficacy	.447	.053	8.409***	.000

Note: t-values > 1.65 (p<0.1); t-values > 1.96** (p<0.05); t-values > 2.57*** (p<0.01)*

Next, the R^2 for *self-assessment* was *.550*, which is the amount of variance in the endogenous variable (*self-assessment*) accounted for by the exogenous variable (*post-programming self-efficacy*) and the mediating variables (*behavioural engagement*, *cognitive engagement*, and *emotional engagement*).

In the final step of determining if mediation exists, the confidence intervals were examined. Table AG.12 presents the lower and upper limits of the confidence intervals for the endogenous variable *self-assessment*. If a zero (0) lies in between the confidence intervals, then no mediation exists. In Table AG.12, the individual indirect effect for *behavioural engagement* and *emotional engagement* contained a 0 in between the confidence intervals. This result confirms that *behavioural engagement* and *emotional engagement* mediated the relationship between *post-programming self-efficacy* and *self-assessment*. Additionally, the total indirect effect also contained a 0 in between the confidence intervals, which confirmed that the *engagement*

constructs cumulatively mediated the relationship between *post-programming self-efficacy* and *self-assessment*.

Table AG.12: Confidence Intervals for endogenous variable programming grade

	Lower	Upper	Mediation?
Total	.157	.373	Yes
Behavioural Engagement	.006	.159	Yes
Cognitive Engagement	-.046	.126	No
Emotional Engagement	.077	.204	Yes

Appendix H: Reliability and Validity

Table AH.1: Reliability and Validity Test

Type of Reliability / Validity test	Assessment Criteria	Model/Tool	Tests/Methods	Reference(s)	Section
Internal consistency *(R)*	• Composite Reliability (CR) > 0.7 • Cronbach's α > .7 • Cronbach's α > .6 acceptable • Cronbach's α > .5 acceptable at early stage of research	Measurement model	PLS Analysis using SEM-PLS – for CR SPSS – for Cronbach's α	Nunally (1978) Fornell & Larcker (1981) Chin (1998a) Hair et al. (2014) Nunnally & Bernstein (1994) Kline (1999) Zikmund et al. (2010)	7.5.2 6.3.2.5
Indicator reliability (*R)*	• Item loadings > .7 • Item loading > .4 and < .7, examine AVE and CR. • Item loading < .4, delete item.	Measurement model	PLS Analysis using SEM-PLS	Hair et al. (2014)	7.5.3
Inter-rater reliability *(R)*	• Cohen's Kappa > .65	Phase 3 Scale Development	SPSS	Jarvenpaa (1989) Moore & Benbasat (1991)	6.2.2
Content Validity *(V)*	• Analyse focus group to confirm/identify engagement constructs • Item Creation • Pre-test of instrument • Card sorting – group items based on similarities	Instrument	• Literature review • Focus groups to determine the engagement constructs • Academics to perform card sorting	Straub (1989) Straub et al. (2004) Moore & Benbasat (1991)	5.3 6.1 6.2

Face Validity *(V)***	• Confirm items fit construct's definition	Instrument	• Expert Review • Pilot study (CFA)	Zikmund et al. (2010)	6.1.1.2 6.2.1.4 6.3.2
Factorial Validity *(V)* ** *(Note: Factorial Validity assesses both convergent and discriminant validity and can also be performed in SEM)*	• Kaiser-Meyer-Olkin (KMO) > .5 (Kaiser, 1974) • Significance of Bartlett's test of Sphericity < .05 • Item loadings > .5 • Item cross loadings < .4	Measurement Model	• EFA using SPSS • CFA using SEM-PLS *(see convergent and discriminant validity below)*	Nunally (1978) Straub (1989) Straub et al. (2004) Field (2013)	6.3.2.7 7.5.1 7.5.3 7.5.4
Convergent Validity *(V)* **	• AVE > .5	Measurement Model	PLS Analysis using SEM-PLS	Straub et al. (2004) Hair et al. (2014)	7.5.3
Discriminant Validity *(V)* **	• Card sorting – group items based on similarities • Indicators should load strongly to assigned construct compared to other constructs • Fornell-Larcker criterion: Square root of the AVE should be more than the correlation with other constructs.	Instrument Measurement Model	Academics to perform card sorting PLS Analysis using SEM-PLS	Moore & Benbasat (1991) Chin (1998) Straub et al. (2004) Hair et al. (2014)	6.2.2 7.5.4

Nomological validity *(V)* **	• Hypothesis testing using bootstrapping • Examine strength of relationships and relate to prior literature (findings) • Variety of methods to measure construct	Research model / Structural Model	• Focus Groups • Bootstrapping using SEM-PLS • Chapter 8: Discussion	Sussmann & Robertson (1986) Straub at al. (2004) MacKenzie et al. (2011)	5.3 7.6 8
Predictive validity *(V) (Criterion-related / concurrent validity)*	• Predictive accuracy (R^2) • Predictive relevance (Q^2)	Structural Model	• Bootstrapping using SEM-PLS • Blindfolding using SEM-PLS	Hair et al. (2014)	7.6
Common Method Bias *(V)*	• EFA – no more than 50% of items should load on one factor • Harman's single factor test – no more than 50% of items should load on one factor	Instrument	EFA using SPSS	Podsakoff & Organ (1986) Straub et al. (2004) Posdakoff et al. (2003)	6.5.3

*** types of construct validity*

(R) – indicates type of reliability test

(V) – indicates type of validity test

Table AH.2: Cross loadings for discriminant validity of measurement model

Construct	Item	Deep Learning	Effort	Enjoyment	Gratification	Help Seeking	Interest	Persistence	Post-programming Self-Efficacy	Pre-programming Self-Efficacy	Programming Grade	Self-Assessment	Surface Learning	Trial and Error
Pre-programming Self-efficacy	BPSE1	0.291	0.199	0.284	0.169	-0.153	0.219	0.265	0.388	0.627	0.263	0.236	-0.223	0.204
	BPSE6	0.319	0.255	0.319	0.218	-0.082	0.224	0.315	0.454	0.649	0.242	0.324	-0.160	0.251
	BPSE12	0.278	0.176	0.269	0.088	-0.107	0.233	0.279	0.388	0.673	0.242	0.220	-0.175	0.234
	BPSE13	0.308	0.229	0.320	0.143	-0.072	0.264	0.305	0.450	0.765	0.253	0.274	-0.129	0.287
	BPSE14	0.288	0.262	0.272	0.120	-0.038	0.228	0.283	0.413	0.779	0.220	0.320	-0.106	0.290
	BPSE15	0.310	0.259	0.316	0.186	-0.059	0.238	0.321	0.417	0.760	0.162	0.344	-0.101	0.227
	BPSE16	0.283	0.139	0.246	0.120	-0.115	0.218	0.229	0.406	0.646	0.142	0.287	-0.155	0.115
	BPSE17	0.247	0.232	0.271	0.093	0.009	0.179	0.209	0.360	0.711	0.103	0.226	-0.099	0.211
	BPSE18	0.308	0.190	0.283	0.090	-0.107	0.249	0.268	0.408	0.725	0.150	0.277	-0.124	0.247
	BPSE21	0.278	0.190	0.251	0.121	-0.095	0.170	0.238	0.412	0.725	0.192	0.229	-0.164	0.271
	BPSE22	0.311	0.190	0.270	0.181	-0.101	0.214	0.231	0.457	0.759	0.245	0.308	-0.210	0.262
	BPSE23	0.291	0.131	0.260	0.208	-0.139	0.167	0.236	0.373	0.682	0.249	0.278	-0.283	0.205
	BPSE24	0.289	0.215	0.243	0.147	-0.065	0.172	0.229	0.402	0.742	0.211	0.328	-0.175	0.232
	BPSE25	0.240	0.161	0.185	0.086	-0.004	0.200	0.201	0.305	0.679	0.119	0.193	-0.131	0.209
Post-programming Self-efficacy	APSE1	0.563	0.419	0.475	0.417	-0.126	0.443	0.583	0.758	0.380	0.281	0.569	-0.185	0.439
	APSE2	0.475	0.441	0.482	0.410	-0.127	0.408	0.512	0.655	0.341	0.283	0.491	-0.138	0.433
	APSE4	0.434	0.384	0.463	0.286	-0.071	0.412	0.424	0.674	0.392	0.281	0.487	-0.191	0.325
	APSE5	0.513	0.482	0.514	0.361	-0.180	0.443	0.485	0.763	0.430	0.335	0.577	-0.231	0.392
	APSE6	0.539	0.377	0.476	0.454	-0.179	0.459	0.476	0.760	0.421	0.352	0.578	-0.219	0.349
	APSE7	0.520	0.350	0.398	0.474	-0.104	0.408	0.480	0.647	0.313	0.252	0.494	-0.163	0.317
	APSE12	0.479	0.457	0.504	0.333	-0.135	0.480	0.509	0.740	0.412	0.286	0.526	-0.189	0.398
	APSE13	0.503	0.412	0.494	0.284	-0.186	0.400	0.477	0.781	0.427	0.336	0.544	-0.200	0.362
	APSE14	0.490	0.426	0.419	0.293	-0.119	0.400	0.454	0.778	0.461	0.349	0.523	-0.212	0.326
	APSE15	0.449	0.370	0.435	0.312	-0.207	0.368	0.397	0.765	0.439	0.325	0.522	-0.279	0.333
	APSE16	0.451	0.360	0.413	0.292	-0.194	0.374	0.411	0.714	0.491	0.266	0.481	-0.233	0.314

Construct	Item	Deep Learning	Effort	Enjoyment	Gratification	Help Seeking	Interest	Persistence	Post-programming Self-Efficacy	Pre-programming Self-Efficacy	Programming Grade	Self-Assessment	Surface Learning	Trial and Error
Post-programming Self-efficacy (contd.)	APSE17	0.528	0.458	0.483	0.365	-0.114	0.469	0.464	0.766	0.443	0.331	0.582	-0.167	0.391
	APSE18	0.553	0.385	0.498	0.313	-0.128	0.448	0.476	0.755	0.483	0.297	0.528	-0.196	0.422
	APSE20	0.464	0.526	0.504	0.336	-0.102	0.458	0.516	0.651	0.367	0.266	0.495	-0.148	0.464
	APSE21	0.461	0.390	0.394	0.271	-0.146	0.370	0.388	0.710	0.434	0.311	0.464	-0.221	0.366
	APSE22	0.502	0.392	0.470	0.349	-0.159	0.432	0.436	0.766	0.449	0.345	0.519	-0.230	0.370
	APSE23	0.496	0.304	0.377	0.384	-0.122	0.279	0.429	0.694	0.379	0.312	0.450	-0.186	0.379
	APSE24	0.497	0.396	0.412	0.333	-0.156	0.321	0.445	0.750	0.428	0.300	0.496	-0.226	0.403
	APSE25	0.461	0.425	0.448	0.270	-0.108	0.435	0.413	0.749	0.390	0.256	0.490	-0.176	0.369
Effort	BEEF2	0.291	0.792	0.28	0.227	0.078	0.360	0.348	0.394	0.142	0.228	0.386	0.061	0.349
	BEEF5	0.285	0.568	0.296	0.146	0.143	0.378	0.443	0.305	0.165	-0.021	0.278	0.093	0.425
	BEEF6	0.312	0.707	0.250	0.258	0.131	0.344	0.412	0.309	0.172	0.040	0.313	0.156	0.454
	BEPA1	0.485	0.787	0.432	0.356	0.030	0.457	0.445	0.532	0.306	0.289	0.496	-0.058	0.405
Help-seeking	BEHS1	-0.059	0.132	-0.104	0.013	0.868	0.001	0.020	-0.170	-0.131	-0.261	-0.084	0.473	0.072
	BEHS3	0.163	0.334	0.080	0.170	0.424	0.150	0.188	0.065	-0.009	-0.072	0.119	0.354	0.237
	BEHS5	-0.001	0.066	-0.132	-0.020	0.820	-0.031	-0.006	-0.147	-0.052	-0.219	-0.091	0.401	0.037
Persistence	BEPE1	0.428	0.459	0.469	0.285	0.011	0.509	0.726	0.473	0.292	0.123	0.338	-0.018	0.463
	BEPE3	0.368	0.310	0.223	0.323	0.103	0.212	0.568	0.297	0.180	0.060	0.273	0.043	0.397
	BEPE4	0.468	0.420	0.397	0.343	0.030	0.377	0.749	0.472	0.260	0.144	0.402	-0.074	0.436
	BEPE5	0.567	0.457	0.535	0.445	-0.081	0.494	0.832	0.572	0.332	0.222	0.459	-0.145	0.501
	BEEF3	0.496	0.361	0.332	0.490	0.008	0.330	0.702	0.416	0.224	0.158	0.389	-0.062	0.407
Deep Learning	CEDL1	0.766	0.383	0.416	0.451	0.001	0.358	0.463	0.477	0.279	0.164	0.430	-0.062	0.398
	CEDL2	0.630	0.354	0.280	0.412	0.064	0.280	0.365	0.383	0.158	0.182	0.394	0.002	0.348
	CEDL4	0.743	0.324	0.419	0.432	-0.116	0.291	0.456	0.502	0.310	0.267	0.378	-0.150	0.367
	CEDL5	0.773	0.391	0.607	0.503	-0.096	0.548	0.555	0.592	0.409	0.243	0.483	-0.181	0.503
	CETE1	0.638	0.307	0.296	0.344	0.037	0.271	0.473	0.423	0.254	0.081	0.328	-0.076	0.362

Construct	Item	Deep Learning	Effort	Enjoyment	Gratification	Help Seeking	Interest	Persistence	Post-programming Self-Efficacy	Pre-programming Self-Efficacy	Programming Grade	Self-Assessment	Surface Learning	Trial and Error
Surface Learning	CESL2	-0.043	0.030	-0.143	0.009	0.376	-0.040	-0.065	-0.237	-0.196	-0.252	-0.106	0.784	0.037
	CESL3	-0.164	0.065	-0.091	-0.178	0.409	-0.013	-0.100	-0.150	-0.149	-0.262	-0.123	0.695	-0.040
	CESL5	-0.180	0.032	-0.209	-0.111	0.393	-0.042	-0.115	-0.263	-0.183	-0.310	-0.202	0.827	0.020
	CESL6	0.106	0.071	0.011	0.097	0.336	0.048	0.171	-0.059	-0.081	-0.160	0.014	0.527	0.185
Trial and Error	CETE2	0.280	0.449	0.228	0.233	0.173	0.280	0.430	0.255	0.133	-0.054	0.258	0.153	0.547
	CETE3	0.442	0.378	0.377	0.371	0.022	0.334	0.460	0.410	0.260	0.133	0.347	0.039	0.827
	CETE5	0.362	0.432	0.272	0.258	0.161	0.287	0.420	0.304	0.174	0.023	0.270	0.058	0.695
	CESL4	0.525	0.427	0.412	0.428	-0.065	0.330	0.512	0.498	0.345	0.198	0.442	-0.058	0.853
Enjoyment	EEEN1	0.458	0.399	0.811	0.449	-0.068	0.641	0.434	0.516	0.246	0.251	0.446	-0.087	0.335
	EEEN2	0.274	0.217	0.685	0.265	-0.124	0.404	0.323	0.349	0.304	0.213	0.351	-0.205	0.297
	EEEN3	0.603	0.427	0.865	0.520	-0.146	0.691	0.543	0.584	0.360	0.278	0.512	-0.157	0.423
Gratification	EEGR1	0.543	0.282	0.445	0.849	-0.070	0.380	0.455	0.441	0.207	0.168	0.446	-0.217	0.368
	EEGR2	0.358	0.103	0.180	0.637	0.086	0.185	0.309	0.216	0.058	-0.017	0.258	0.107	0.301
	EEGR4	0.382	0.351	0.508	0.685	0.078	0.544	0.365	0.321	0.134	0.090	0.320	0.018	0.340
	EEGR5	0.543	0.330	0.463	0.878	-0.047	0.453	0.459	0.414	0.171	0.130	0.468	-0.089	0.384
Interest	EEIN1	0.367	0.434	0.475	0.405	0.134	0.699	0.371	0.290	0.108	0.005	0.331	0.112	0.255
	EEIN3	0.521	0.482	0.698	0.472	-0.067	0.885	0.528	0.570	0.320	0.184	0.513	-0.091	0.416
	EEIN4	0.343	0.424	0.644	0.408	-0.056	0.872	0.436	0.474	0.262	0.172	0.434	-0.040	0.321
Self-assessment	SA1	0.541	0.524	0.550	0.403	-0.107	0.511	0.491	0.689	0.363	0.417	0.874	-0.178	0.409
	SA2	0.463	0.428	0.467	0.433	-0.102	0.420	0.408	0.604	0.357	0.272	0.871	-0.154	0.404
	SA5	0.396	0.345	0.337	0.413	-0.049	0.358	0.389	0.423	0.220	0.148	0.714	-0.086	0.310
Programming Grade*	ProgGrade	0.271	0.230	0.314	0.140	-0.287	0.166	0.207	0.416	0.282	1.000	0.355	-0.354	0.133

**Single-item construct*

Appendix I: Reliability and Validity of lower-order pre- and post-programming self-efficacy constructs

Table AI.1: Reliability of lower-order pre- and post-programming self-efficacy constructs

Higher-order Construct	Lower-order construct	Indicator	Loadings	AVE	CR	Cronbach's α
Pre-programming Self-Efficacy	**General self-efficacy**	*BPSE1*	.760	.544	.877	.832
		BPSE2	.711			
		BPSE3	.684			
		BPSE4	.756			
		BPSE5	.755			
		BPSE6	.756			
	Independence and Persistence	*BPSE7*	.639	.500	.845	.795
		BPSE8	.651			
		BPSE9	.717			
		BPSE10	.693			
		BPSE11	.667			
		BPSE12	.646			
		BPSE13	.613			
	Complex Programming Tasks	*BPSE14*	.819	.619	.890	.845
		BPSE15	.827			
		BPSE16	.722			
		BPSE17	.789			
		BPSE18	.771			
	Self-regulation	*BPSE19*	.891	.774	.872	.708
		BPSE20	.868			
	Simple Programming Tasks	*BPSE21*	.803	.644	.900	.861
		BPSE22	.857			
		BPSE23	.789			
		BPSE24	.814			
		BPSE25	.745			
Post-programming Self-Efficacy	**General self-efficacy**	*APSE1*	.839	.613	.905	.873
		APSE2	.753			
		APSE3	.753			
		APSE4	.742			
		APSE5	.810			
		APSE6	.797			
	Independence and persistence	*APSE7*	.732	.538	.891	.859
		APSE8	.692			
		APSE9	.757			
		APSE10	.722			

Higher-order Construct	Lower-order construct	Indicator	Loadings	AVE	CR	Cronbach's α
	Complex Programming Tasks	*APSE11*	.806	.671	.911	.877
		APSE12	.714			
		APSE13	.708			
		APSE14	.820			
		APSE15	.835			
		APSE16	.796			
		APSE17	.831			
		APSE18	.811			
	Self-regulation	*APSE19*	.894	.816	.899	.775
		APSE20	.913			
	Simple Programming Tasks	*APSE21*	.813	.672	.911	.878
		APSE22	.848			
		APSE23	.805			
		APSE24	.820			
		APSE25	.812			

Table AI.2: Convergent validity of lower-order programming self-efficacy constructs

Item	General Self-efficacy	Independence and persistence	Complex programming tasks	Self-regulation	Simple programming tasks
BPSE1	0.760	0.528	0.526	0.265	0.449
BPSE2	0.711	0.471	0.485	0.387	0.426
BPSE3	0.684	0.421	0.358	0.345	0.329
BPSE4	0.756	0.439	0.495	0.338	0.478
BPSE5	0.755	0.480	0.472	0.330	0.447
BPSE6	0.756	0.542	0.524	0.276	0.507
BPSE7	0.419	0.667	0.323	0.206	0.281
BPSE8	0.322	0.646	0.253	0.165	0.218
BPSE9	0.243	0.613	0.202	0.101	0.169
BPSE10	0.426	0.639	0.345	0.222	0.315
BPSE11	0.301	0.651	0.275	0.185	0.225
BPSE12	0.568	0.717	0.563	0.302	0.513
BPSE13	0.565	0.693	0.692	0.308	0.577
BPSE14	0.574	0.548	0.819	0.360	0.600
BPSE15	0.566	0.550	0.827	0.393	0.582
BPSE16	0.434	0.477	0.722	0.309	0.509
BPSE17	0.475	0.430	0.789	0.312	0.562
BPSE18	0.496	0.474	0.771	0.398	0.602
BPSE19	0.375	0.324	0.440	0.891	0.397
BPSE20	0.392	0.281	0.352	0.868	0.358
BPSE21	0.497	0.463	0.582	0.421	0.803

Item	General Self-efficacy	Independence and persistence	Complex programming tasks	Self-regulation	Simple programming tasks
BPSE22	0.491	0.460	0.620	0.345	0.857
BPSE23	0.475	0.448	0.522	0.289	0.789
BPSE24	0.514	0.462	0.605	0.339	0.814
BPSE25	0.427	0.362	0.587	0.325	0.745

Table AI.3: Cross loadings for discriminant validity of the indicators of pre-programming self-efficacy

Items	General self-efficacy	Independence and Persistence	Complex Programming Tasks	Self-regulation	Simple Programming Tasks
APSE1	0.839	0.642	0.617	0.439	0.619
APSE2	0.753	0.524	0.548	0.492	0.491
APSE3	0.753	0.559	0.474	0.480	0.473
APSE4	0.742	0.517	0.580	0.445	0.527
APSE5	0.810	0.587	0.667	0.505	0.621
APSE6	0.797	0.691	0.642	0.439	0.615
APSE7	0.637	0.806	0.525	0.334	0.493
APSE8	0.410	0.714	0.347	0.188	0.336
APSE9	0.409	0.708	0.363	0.256	0.352
APSE10	0.578	0.732	0.523	0.383	0.456
APSE11	0.393	0.692	0.340	0.252	0.340
APSE12	0.675	0.757	0.653	0.466	0.573
APSE13	0.634	0.734	0.722	0.490	0.659
APSE14	0.646	0.582	0.820	0.501	0.650
APSE15	0.648	0.608	0.835	0.479	0.627
APSE16	0.584	0.576	0.796	0.376	0.598
APSE17	0.619	0.586	0.831	0.522	0.650
APSE18	0.592	0.559	0.811	0.534	0.654
APSE19	0.512	0.409	0.512	0.893	0.498
APSE20	0.560	0.464	0.553	0.913	0.574
APSE21	0.533	0.483	0.628	0.480	0.813
APSE22	0.599	0.557	0.667	0.550	0.848
APSE23	0.560	0.553	0.582	0.412	0.805
APSE24	0.626	0.549	0.650	0.495	0.820
APSE25	0.615	0.522	0.653	0.497	0.812

Table AI.4: Cross loadings for discriminant validity of the indicators of post-programming self-efficacy

Lower-order constructs	Complex programming tasks	General self-efficacy	Independence and persistence	Self-regulation	Simple programming tasks
Complex programming tasks	0.787				
General self-efficacy	0.651	0.738			
Independence and persistence	0.633	0.653			
Self-regulation	0.452	0.436	0.205	0.879	
Simple programming tasks	0.727	0.600	0.363	0.430	0.803

Table AI.5: Fornell-Larcker criterion for discriminant validity of the indicators of pre-programming self-efficacy

Lower-order constructs	Complex programming tasks	General self-efficacy	Independence and persistence	Self-regulation	Simple programming tasks
Complex programming tasks	0.819				
General self-efficacy	0.755	0.783			
Independence and persistence	0.711	0.752	0.734		
Self-regulation	0.590	0.594	0.484	0.903	
Simple programming tasks	0.777	0.717	0.650	0.595	0.820

Appendix J: Hypothesis testing using Bootstrapping (with pre-programming self-efficacy)

Table AJ.1: Hypothesis testing using Bootstrapping (with pre-programming self-efficacy)

Relationship	Path Coefficient	Standard Error	t-value	*p*-value	Decision
Pre-programming self-efficacy -> Help-seeking	*-.111*	*.054*	*2.072***	*.039*	*Not Supported*
Pre-programming self-efficacy -> Effort	.289	.048	6.024***	.000	Supported
Pre-programming self-efficacy -> Persistence	.365	.040	9.142***	.000	Supported
Pre-programming self-efficacy -> Deep Learning	.410	.043	9.553***	.000	Supported
Pre-programming self-efficacy -> Surface Learning	-.222	.049	4.499***	.000	Supported
Pre-programming self-efficacy -> Trial and Error	.332	.043	7.666***	.000	Supported
Pre-programming self-efficacy -> Interest	.301	.042	7.128***	.000	Supported
Pre-programming self-efficacy -> Enjoyment	.386	.042	9.216***	.000	Supported
Pre-programming self-efficacy -> Gratification	.200	.043	4.654***	.000	Supported

Note: t-values > 1.65 (p<0.1); t-values > 1.96** (p<0.05); t-values > 2.57*** (p<0.01)*

Appendix K: Selection of indicators of engagement in introductory programming courses

Table AK.1: Selection of indicators of engagement in introductory programming courses

Dimension	Proposed Construct	Informing References (programming)	Informing References (other disciplines)	Findings	Decision
Behavioural	Effort	McKinney & Denton (2004); Ventura (2005)	Bandura (1977); Diseth et al. (2010); Dupeyrat & Mariné, (2005); Liu, Cheng, Chen & Wu (2009); McClure et al. (2011); Pintrich & Schunk (1996); Weiner (1985); Zimmerman (2000)	Significant correlations observed between effort and programming performance. Self-efficacy has an effect on effort. Effort has an effect on academic achievement	Accept
	Persistence	White (2004)	Bandura (1977); Brown et al. (2008); Glastra, Hake & Schedler (2004); Pintrich & Schunk (1996); Zimmerman (2000);	Persistent students had a higher GPA in Information Technology. Self-efficacy has an effect on persistence. Persistence influences performance.	Accept
	Attention	Bednarik & Tukiainen (2004); Sillitti, Succi & Vlasenko (2012);	Deviney, Crawford & Elder (2012)	Examines student's attention span and engagement in a course. Attention could be embedded into the participation indicator. Reason to reject: No evidence to support the link between attention and performance.	Reject

Dimension	Proposed Construct	Informing References (programming)	Informing References (other disciplines)	Findings	Decision
Behavioural	Participation	Shaw (2013)	Appleton et al. (2006); Christenson & Reschly (2012); Galyon et al. (2012)	Participation in online programming forum significantly improved learning scores. Participation is an indicator of behavioural engagement. Self-efficacy significantly predicted participation in class discussion.	Accept
	Practise	Hassinen & Mäyrä (2006)	-	Need extensive practise to be familiar with programming concepts. There is a gap in literature on the importance of practise in programming and in other educational disciplines. Reason to reject: No evidence in literature on its importance and it is possible that practise may be seen as an effort-based engagement behaviour or a deep learning strategy.	Reject

Dimension	Proposed Construct	Informing References (programming)	Informing References (other disciplines)	Findings	Decision
Behavioural	Help-seeking	-	Bembenutty & White (2013); Karabenick & Newman (2006); Karabenick (2003); Nelson-Le Gall & Glor-Scheib (1985); Ryan & Shin (2011); Ryan & Pintrich (1998)	Self-efficacy and adaptive help-seeking is positively associated with course grade. Adaptive help-seekers had higher grades while avoidant help-seekers performed poorly. Help-seeking improves performance in a task. Academic self-efficacy positively related to adaptive help-seeking and negatively related to avoidant help-seeking. A positive relationship exists between self-efficacy and instrumental help-seeking.	Accept
	Attendance	These indicators were derived from the engagement models discussed in Chapter 2, Section 2.3.2. Reason to reject: These indicators may better reflect behavioural engagement in schools compared to learning at the tertiary level which emphasises independent and self-directed learning.			Reject
	Credit options				Reject

Dimension	Proposed Construct	Informing References (programming)	Informing References (other disciplines)	Findings	Decision
Cognitive	Deep Learning	de Raadt et al. (2005); Hughes & Peiris (2006); Simon et al. (2006)	Diseth et al. (2010); Fenollar, Román & Cuestas (2007); Miller et al. (1996); Phan (2011); Pintrich & De Groot (1990); Schunk & Mullen (2012); Yip (2012)	Strong negative correlation between surface learning and academic achievement in programming. Weak positive correlation between deep learning and academic achievement in programming. Learning approaches had the strongest correlations to success in programming. Self-efficacy influences cognitive engagement. Self-efficacious learners are likely to use learning strategies that they believe will lead to success. Positive correlations between self-efficacy, deep learning and academic achievement and negative correlations between self-efficacy surface learning and academic achievement. Different study strategies correlate with performance.	Accept
	Surface Learning				Accept

Dimension	Proposed Construct	Informing References (programming)	Informing References (other disciplines)	Findings	Decision
Cognitive	Trial and Error	Dorn & Guzdial (2010); Edwards (2004)	Carlson & Skaggs (2000); Ebrahimi (2012); Matzat & Sadowski (2012)	Trial and error appears to be a strategy for debugging errors in programming code and for troubleshooting problems. Reason to consider: Some evidence in literature of students using a trial and errors strategy and appears to fit the definition of engagement.	Likely
	Self-regulation	Reason to consider: Self-regulation and metacognition are important but may require a large scale study by its own since self-regulation is a model which requires multiple indicators to examine the self-regulated strategies of a student.			Likely
	Metacognition				Likely
	Value of learning	Reason to reject: Value of learning, strategizing, and relevance of learning are excluded from this study as this study assumes that students are able to see the value in their learning and understand the relevance of their learning based on their decision to undertake a field of study at the tertiary level of education.			Reject
	Strategizing				Reject
	Relevance of learning				Reject

Dimension	Proposed Construct	Informing References (programming)	Informing References (other disciplines)	Findings	Decision
Emotional	Affect	Rodrigo et al. (2009); Teague & Roe (2008)	Sheard, Carbone & Hurst (2010)	Affect refers to emotions or affective experiences when learning (Linnenbrink & Pintrich, 2003). This indicator is reflective of various positive and negative emotions. Anxiety, delight, surprise, frustration, and neutrality did not predict programming performance (Rodrigo et al., 2009). Reason to reject: The other indicators proposed in this dimension may collectively determine affect.	Reject
	Interest	Akbulut & Looney (2007); McKinney & Denton, (2004); Sheard, Carbone & Hurst (2010); Wiedenbeck et al.(2007)	Bandura (1977); Bye, Pushkar & Conway (2007); Linnenbrink & Pintrich, 2003; Rottinghaus, Larson, and Borgen (2003); Silvia (2003)	Interest is significantly correlated to programming performance and a strong predictor of motivation to learn. Students rated high interest in topics such as programming and computer networks. Positive relationship between interest and programming performance.	Accept

Dimension	Proposed Construct	Informing References (programming)	Informing References (other disciplines)	Findings	Decision
Emotional	Enjoyment	Bishop-Clark, Courte, Evans & Howard (2007); Liebenberg, Mentz & Breed (2012)	Frenzel, Thrash, Pekrun & Goetz, (2007); Mills, Pajares & Herron (2007); Pekrun, Goetz, Titz & Perry (2002); Chen & McGrath (2003); Pekrun et al. (2004); Putwain, Sander & Larkin (2013)	Using Alice increases enjoyment in programming. Pair programming increases joy in programming. Enjoyment correlated to academic achievement. Positive relationship between self-efficacy and enjoyment and between enjoyment and performance. High self-efficacy results in pleasant emotions such as enjoyment.	Accept
	Value	These indicators were derived from the engagement models discussed in Chapter 2, Section 2.3.2. Reason to reject: These indicators may have a stronger impact on the student's perception of the educational institution that they are attending rather than the programming performance of the novice programmer.			Reject
	Belonging				Reject
	Identification with school				Reject
	School membership/ connectedness				Reject