Identification and Evaluation of Predictors for Learning Success and of Models for Teaching Computer Programming in Contemporary Contexts

By

Nick Day

A thesis submitted in partial fulfilment of the University's requirements for the Doctor of Philosophy

March 2020

Buckinghamshire New University

Coventry University

COPYRIGHT

This copy of the thesis has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with its author under the terms of the United Kingdom Copyright Acts. No quotation from the thesis and no information derived from it may be published without proper acknowledgement.

Abstract

Introductory undergraduate computer programming courses are renowned for higher than average failure and withdrawal rates when compared to other subject areas. The closer partnership between higher education and the rapidly expanding digital technology industry, as demonstrated by the establishment of new Degree Apprenticeships in computer science and digital technologies, requires efficient and effective means for teaching programming skills. This research, therefore, aimed to identify reliable predictors of success in learning programming or vulnerability to failure. The research also aimed to evaluate teaching methods and remedial interventions towards recommending a teaching model that supported and engaged learners in contemporary contexts that were relevant to the workplace.

Investigation of qualifications designed to prepare students for undergraduate computer science courses revealed that A-level entrants achieved significantly higher programming grades than BTEC students. However, there was little difference between the grades of those with and those without previous qualifications in computing or ICT subjects.

Analysis of engagement metrics revealed a strong correlation between extent of co-operation and programming grade, in contrast to a weak correlation between programming grade and code understanding. Further analysis of video recordings, interviews and observational records distinguished between the type of communication that helped peers comprehend tasks and concepts, and other forms of communication that were only concerned with completing tasks.

Following the introduction of periodic assessment, essentially converting a single final assessment to three staged summative assessment points, it was found that failing students often pass only one of the three assignment parts. Furthermore, only 10% of those who failed overall had attempted all three assignments. Reasons for failure were attributed to 'surface' motivations (such as regulating efforts to achieve a minimum pass of 40%), ineffective working habits or stressful personal circumstances rather than any fundamental difficulty encountered with subject material.

A key contribution to pedagogical practice made by this research is to propose an 'incremental' teaching model. This model is informed by educational theory and empirical evidence and comprises short cycles of three activities: presenting new topic information, tasking students with a relevant exercise and then demonstrating and discussing the exercise solution. The effectiveness of this model is evidenced by increased engagement, increased quiz scores at the end of each teaching session and increased retention of code knowledge at the end of the course.

Contents

List of Figures

List of Tables

Acknowledgements

Firstly, I would like to thank Dr Richard Mather and Dr Kevin Maher, in addition to advisor Richard Jones and the entire Bucks Computing Department for their invaluable guidance and support during this project. I owe a special mention to Indrachapa Bandara (IB) for providing access to research journals and to Mike Everett for liaising with our partner college in Sri Lanka on my behalf.

I'd like to thank the Bucks Research Department (Laura, Anne and Mel) and the community of fellow PhD students and researchers for their support. I'm also grateful to colleagues such as Mike Mousely for teaching the PGCert; Jason Schaub for teaching Research Methods and Dr Vasos Pavlika for the opportunity to work at Oxford University during this time.

Many thanks are also owed to the students who participated in my studies, offering their time and comments to further this research.

I wouldn't have been able to complete this work without my parents who provided for me and kindly allowed me to live with them during this project. I also want to dedicate this work to my grandparents (Eileen and Bernard Teed) who sadly passed in June 2018 and October 2019 respectively.

I'm grateful to many friends who journeyed with me; my old school friends, university friends, and church family. Special mention is owed to Dr Peter Brewer for his initial advice; John Chambers for proof reading and fruitful discussions; Jennie Liebenberg for additional proof reading; as well as the Hollier's, Gorst's, Bigby's, Silk and Miro Mateev, Tim and Chloe Grace, Pao Mbewe, Jen Marris and Andrew Henley for their valuable support.

And finally, glory to God, Jesus Christ, who has been with me during the highs and lows of this journey. Thank you for all you've taught me and helped me to learn about myself.

Author’s Declaration

I declare that this thesis and the work presented in it are my own and have been generated by me as the result of my own original research. I confirm that:

1. This work was done wholly or mainly while in candidature for a research degree at this University.

2. Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.

3. Where I have consulted the published work of others, this is always clearly attributed.

4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.

5. Where elements of this work have been published or submitted for publication prior to submission, this is identified and references given at the end of the thesis.

6. This thesis has been prepared in accordance with Coventry University and Buckinghamshire New University regulations.

7. I confirm that if the submission is based upon work that has been sponsored or supported by an agency or organisation that I have fulfilled any right of review or other obligations required by such contract or agreement.

Nick Day

Publications arising from this thesis

Mather, R., Day, N., Jones, R., Lusuardi, C., Maher, K., and Dexter, B. (2015) Canonical explorations of ‘TEL’ Environments for Computer Programming. *The European Conference on Technology in the Classroom 2015: Official Conference Proceedings* [online] 265-282. available from <http://iafor.org/archives/proceedings/ECTC/ECTC2015_proceedings.pdf> [November 2015]

Day, N., Mather, R., Jones, R., Lusuardi, C., Maher, K., and Dexter, B. (Upcoming) Can Educational Profiles Explain Achievement When Learning Computer Programming Using a Technology-Enhanced Environment? *Research in Learning Technology*

Chapter 1 Introduction

This chapter provides a rationale for the research undertaken and outlines three trends including: (i) the growing digital technology industry in the UK; (ii) recent developments and challenges in programming education; and (iii) increased tuition fees and mass participation in Higher Education (HE). These trends underpin the research aim and are addressed in the sections summarised below.

Section 1.1 describes the growth of the UK's digital technology industry and its contributions to the UK economy. Expansion of this industry is exponentially creating employment opportunities for programmers and software engineers. However, businesses are continuing to report a shortfall of adequately skilled employees to meet market demand.

Section 1.2 discusses the introduction of the new national computing curriculum for compulsory education and its effectiveness in motivating further study. This section also discusses how sustained non-continuation rates reported for undergraduate computer science (and programming related) courses are impacting on graduate numbers.

Section 1.3 reviews recent statistics concerning computer science degree applications, and patterns of increasing student participation in UK HE. This section discusses the impact of mass student participation and increased tuition fees on expectations and motivations towards learning.

Section 1.4 describes how the three contextual issues, (i), (ii) and (iii), underpin the need for this research and a priority to respond to issues surrounding the delivery of programming courses. This section introduces the need to identify reliable predictors of student performance to inform effective interventions, thereby improving retention, learning experiences and teaching quality.

Section 1.5 states the aim and objectives of the research.

1.1 UK's growing digital technology industry

The digital sector is one of the fastest growing sectors in the UK, and one of the strongest globally (Bounds & O'Connor 2015). The number of digital technology companies formed in the UK doubled between 2010 and 2013 (Tech City 2015). The 'tech' industry consists of companies that specialise in IT, software and services; telecoms; manufacture of IT and telecoms; retail; and computer games. This sector contributed £91.1 billion to the UK economy in 2014 (Tech Partnership 2015), over half of which was from IT software and services alone (£53.1 billion). The combined digital and tech sector contributed almost £184 billion in 2017 (Ismail 2018).

The ubiquity of digital technology has also led to the 'digitisation' of the whole economy (House of Lords 2015: 6). For example, integrating technology within the automotive industry has led to the innovation of driverless cars and autonomous vehicles (Schiller 2015; Rojas et al. 2011). Surgeons are now able to print (3D) organs and artificial limbs (House of Lords, 2015: 6). Analysis of 'Big Data' is transforming consumer and public services (Virmani 2017). Even the agricultural industry is benefiting from 'smart greenhouses' and more accurate monitoring of growth cycles (Ryan 2016). The internet itself is now relied upon heavily for communication, entertainment and e-commerce (Frey 2017; Lobel 2016; McKetta 2017; Statista 2013). Researchers refer to this era, characterised by personal computing and the internet, as the 'fourth industrial revolution' (Schwab 2016), and the 'Second Machine Age' (Brynjolfsson & McAfee 2014), marking a significant transition from the technology of the 20th century (the 'First Machine Age').

1.1.1 Job creation in the computing industries

This next stage of industrial revolution points towards another shift in the working climate, evident by the creation of unique jobs in digital technologies. The UK Commission for Employment and Skills (2015: 8) highlights emerging areas such as "cyber security; convergence of content across platforms; mobile and cloud computing; big data and analytics; the automation of routine tasks; new applications of social media; and new business models and collaborative platforms". It is anticipated that the future evolution of digital technologies will continue to create unique job roles (World Economic Forum 2016: 3; Frey 2011).

Research groups have attempted to model the numbers of digitally literate workers that are required each year to sustain current industry expansion. The UK Commission for Employment and Skills predicts that some 520,000 jobs requiring highly skilled computer science graduates will be created between 2012 and 2022 (The UK Commission for Employment and Skills 2015: 72-3). Data from Bureau for Labor Statistics (BLS, 2014: 7) predicts that USA's software engineering profession is "projected to experience 23% increase in employment openings between 2012 and 2022".

The Tech Nation (2018: 7) reported that digital tech sector employment rose 13% from 2014 to 2017. The Global IT research organisation, CompTIA, noted an increase in UK IT job advertisements from 1.19 million in 2015, to 1.22 million in 2016, accounting for 13% of total UK job postings (CompTIA 2017), with further growth of 6% to 1.3 million IT jobs advertised in 2017 (CompTIA 2018a). The most advertised job categories in 2017 were programmers and software development professionals, along with IT specialist managers and telecommunication professionals (CompTIA 2018a).

1.1.2 The shortage of skills in digital technologies

Despite the continuing expansion of the tech industry, and the associated increase in jobs, there are concerns that many candidates lack adequate skills to fulfil these roles. Analysts predict that almost 90% of jobs will require "some kind of digital proficiency" over the next two decades (House of Commons Science and Technology Committee 2016). Yet some 23% of adults (12.6 million) in the UK have been identified as lacking the rudimentary digital skills needed to complete basic tasks online, such as using search engines and sending emails, as well as resolving problems with their smartphones and related devices (House of Commons Science and Technology Committee 2016; Ipsos MORI, 2015: 5). A 2017 survey found that three in four businesses now report a shortage of digitally skilled workers (British Chambers of Commerce 2017). This recruitment 'crisis' is stifling industry growth (Horton 2018) and reducing the digital industry's contribution to the economy (House of Commons Science and Technology Committee 2016). Researchers from the working group 'Strategy&' (2015: 8) estimate that the digital skills gap cost the UK £63 billion in GDP (gross domestic product) in 2011.

In addition to a widespread deficit in basic digital skills, research indicates a shortage of graduates and employees with advanced and specific skills in 'STEM' subjects (science, technology, engineering, mathematics), all of which underpin computer science and computational thinking. A report by Engineering UK (2016: 9) found that 46% of those businesses surveyed cite a shortage of STEM graduates as being a "key factor in being unable to recruit appropriate staff". Research by the Social Market Foundation (Broughton 2013: 35) estimates there is an annual shortfall of 40,000 STEM skilled workers in the UK. Research reported in the 2017 edition of Engineering UK (2017: 192) similarly estimates a yearly deficit of 60,000 engineers. The World Economic Forum has placed 'Complex problem solving' at the top of its list of ten most desired skills for graduates and employees in 2020 (World Economic Forum 2010: 22). Problem-solving and communication skills frequently appear in the top five desired skills by UK job advertisements (CompTIA 2018b).

1.2 The current state and challenges in computer science education

The proliferation of employment opportunities, coupled with a shortage in appropriately skilled graduates and employees, has highlighted the importance of improving the learning and teaching of key skills in preparatory stages. These skills and knowledge shortfalls have also instigated reviews of compulsory schooling and undergraduate level study to identify issues and possible remedial measures at each stage of education.

1.2.1 The new 2014 computing school curriculum

Industry leaders criticised the previous ICT secondary school curriculum for being inadequate to meet the future workforce requirements of the 'tech' industry (Cellan-Jones 2011; Computing at School 2012: 11; Department for Business Innovation and Skills 2011; Gove 2012; Livingstone & Hope 2011; Livingstone 2012). In addition, the pre-2014 ICT curriculum was widely regarded to be uninspiring by secondary school pupils, and a disincentive to pursue subjects related to computing beyond compulsory level education (Department for Business Innovation and Skills 2011).

The Secretary of State for Education, Michael Gove (between 2010 and 2014), worked with the British Computer Society (BCS) in collaboration with Computing at School (CAS) and the Royal Academy of Engineering to develop a new computing curriculum (Computing at School 2012; Royal Society 2012). This was implemented in September 2014 (Cellan-Jones 2014; Design Commission 2014). The new curriculum replaced the previous ICT curriculum in primary and secondary schools to more actively promote software engineering and computer science as attractive subjects for study and as career destinations (Department for Business Innovation and Skills 2015; Hlubinka 2015).

At Key Stage 1 (KS1) children aged 5-6 are now taught algorithms and how to create and debug simple programs. At KS2 (ages 7-11) children are introduced to fundamental programming structures (sequence, selection and iteration), the use of variables for storing information and the different forms of input and output devices (Department for Education 2013a). They are also required to apply algorithms for purposes of logical reasoning and to understand their use for data searching and sorting purposes. More advanced KS3 (ages 11-14) introduces Boolean logic and requirements to understand two or more programming languages and concepts of abstraction. KS4 (ages 14-16) is more concerned with the creative and efficient application of KS1-3 learning to solve challenging problems using more complex scenarios (Department for Education 2013b).

This new curriculum places greater emphasis on those fundamental skills, such as computational thinking and programming (Computing at School & NAACE 2014: 7) that are particularly valued by industry (Shadbolt 2016: 5). In addition, organisations such as *Code.org*, *Code Club* and *CoderDojo* are assisting schools in the delivery of voluntary extra-curricular code clubs to encourage children to learn computer programming skills (code.org 2016; De Kock & Gardner 2015; Geere 2012).

1.2.2 Non-continuation rates in undergraduate computer science degree courses

The non-continuation rates of students on HE computer science courses are reported to be significantly greater than other subjects. Research by the Higher Education Funding Council for England (HEFCE) found that 12% of young students (18-20 year olds) and 17% of mature students (21 or older) who enrolled on computer science courses in 2010/11 were no longer studying a year later (HEFCE 2013: 7). This was almost double the average non-continuation rate for other subjects: 6.3% for young students; 11.6% for mature students (HEFCE 2013: 2). The volume of research investigating poor progression in computer science degree programmes over many decades suggests this is a globally and historically pervasive problem (Bennedsen & Caspersen 2007; Peterson & Howe 1979). First year introductory programming courses have been a specific focus of research since the 1980s (Sleeman 1986).

Contemporary studies suggest that the percentage of students dropping out from or failing an introductory programming course is between 30% and 60% (Bennedsen & Caspersen 2007; Bornat 2011; Dehnadi & Bornat 2006; Robins 2010). However, Watson and Li (2014) commented that sources for these figures tended to be anecdotal and lacked quantitative evidence. They cite Bennedsen and Caspersen's study (2007) as the only attempt to properly quantify failure rates, which, even then, was limited by a low participation rate among the targeted education establishments (only 63 complete responses – most of which were from US institutions - were received from the 497 invited). The authors calculated aggregate percentages (where larger classes have more weight) and average percentages (where classes of all sizes had equal weighting) but found minimal difference between the two values. The pass rate ranged from 69% to 72%. They also found that fewer than 3% of students never attended a class; students who abort studies ranged from 12% to 15%; and those who failed ranged from 11% to 13% (Bennedsen & Caspersen 2007: 33).

Watson and Li's (2014: 43) large scale meta-analysis of 161 pieces of research, and this author's analysis of data from Buckinghamshire New University (Bucks 2015a; Bucks 2015b; Bucks 2016a; Bucks 2016b), for two academic years 2014/15 and 2015/16, also reveal that approximately one third of students do not pass the introductory programming course. Watson and Li (2014) noted that the 'drop-out' rate had not improved significantly from 1979 to 2013, and that rates were unrelated to the particular programming language being taught. Both recent small- and large-scale studies consistently report that approximately one third of students do not progress beyond introductory programming courses (Bennedsen & Caspersen 2007). Such non-completion of computer science degrees represents lost potential talent and consequent shortfalls in the numbers of highly skilled STEM graduates needed by industry (The UK Commission for Employment and Skills 2015).

Research investigating the specific reasons for non-continuation in introductory programming and computer science degree courses suggests that some students fail or withdraw due to extraneous factors and personal reasons that hindered their studies (Bennedsen & Caspersen 2008). Others, however, drop out due to weak motivation (Kinnunen & Malmi 2006), an inability to attend lectures and complete assignments (Nikula et al. 2011: 12) and/or difficulties in learning key concepts and completing assignments (Gordon 2016: 6). Whilst educators have limited influence over personal circumstances that affect a student's ability to study, they might be able to mitigate impacts by modifying course structure, content and assessments (Margulieux et al. 2012).

Specific pedagogical interventions for improving the learning and teaching of computer programming and elevating progression rates are further discussed in Sections 1.4 and 2.3.

1.3 Trends in Higher Education

1.3.1 Increased Higher Education participation rates in the UK

The increase in student applications to study computer science degrees reflects the substantial growth in undergraduate and postgraduate student numbers during the past 60 years. Estimates calculate that less than 5% of the UK's 18- and 19-years olds went to university in the 1950s (Times Higher Education 2013) compared with almost 50% of 18-30 year olds who attended university during the period from about 2010 to 2015 (Department for Business Innovation and Skills 2014; Department for Education 2016).

Figure 1.1 illustrates trends in student participation in HE in the UK. The steady rise throughout the 1950s to the 1970s resulted from increasing prosperity and educational reform following the end of the Second World War (Edwards 1982: 50; Robbins 1963). Attitudes towards higher education became more inclusive during the 1960s as government figures believed that "Higher Education should be available to all who are qualified by ability and attainment to pursue it" (Robbins 1963; Department for Business Innovation and Skills 2016). The student population effectively doubled from 7% in 1963 to 15% in 1970 (Walford 1991) but participation rates stagnated during the 1970s and 1980s, largely due to funding cuts motivated by the economic recession at the time (Bathmaker 2003: 177; Edwards 1982: 193). However, participation rates later rose again from 15% in 1988 to 30% in 1992. This growth was stimulated by an improved economic climate and because many polytechnics were granted university status in 1992 (Bathmaker 2003: 177; Huw 1997). Participation rates then remained at around 33% throughout the 1990s and early 2000s, largely due to the introduction of a cap on student recruitment (Bathmaker 2003: 178; National Committee of Inquiry into Higher Education 1997) and the introduction of tuition fees in 1998 (Alley & Smith 2004). Redefining the participation rate to include mature students and part-time learners (all students aged 17-30) led to a further apparent increase in student numbers (Times Higher Education 2013). The economic recession of 2008 also motivated more school leavers and mature students to attend university to improve their employability prospects (Curtis 2009; Lipsett 2009; Universities UK 2010: 5).

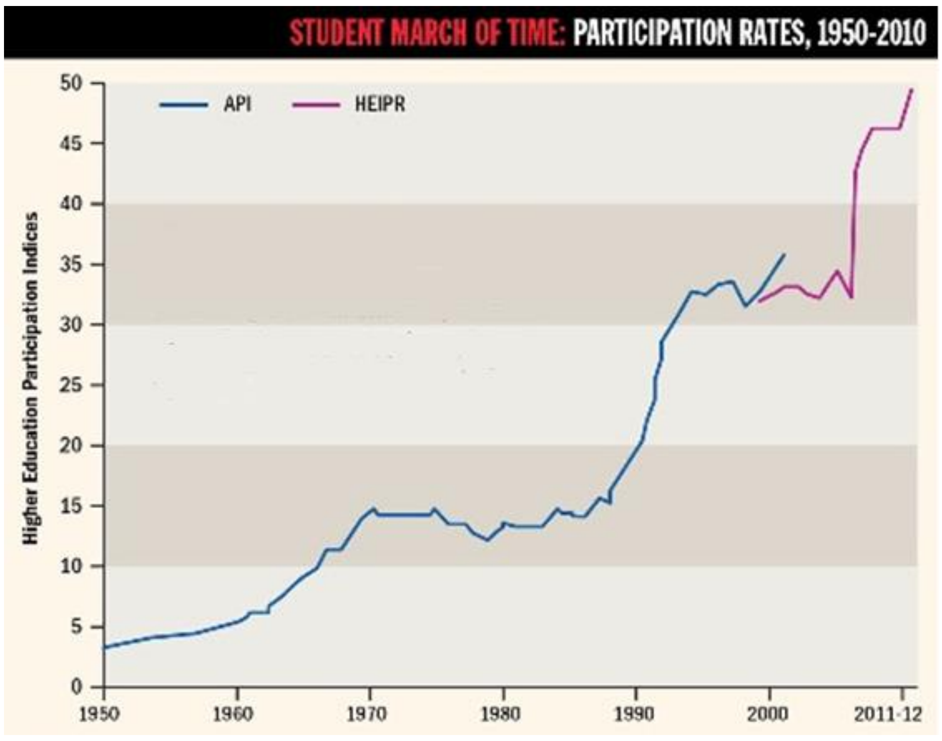


Figure 1.1. Student March of Time: Participation Rates 1950-2010 (Times Higher Education 2013). Notes: (1) The source for this data is the Department for Business, Innovation and Skills/Office for National Statistics (2011-12 figures were provisional); (2) API (Age Participation index): the number of home-domiciled under-21 initial entrants to UK higher education, expressed as a proportion of their average 18- and 19-year-old population; and (3) HEIPR: the Higher Education Initial Participation Rate is the sum of the initial participation rates over the ages of 17-30 for English-domiciled first-time participants in the UK higher education.

Figure 1.2 illustrates that participation rates rose steadily from 42% in 2006/07 to a peak of 49% in 2011/12 (Department for Education 2016). However, the subsequent decline in 2012/13 is largely attributed to increased tuition fees for that academic year (Independent Commission on Fees 2015). Despite this initial decline, participation rates have since returned to levels similar to those before 2012/13, suggesting that fee increases have become less of a disincentive to university entrance (Independent Commission on Fees 2015; UCAS 2014).

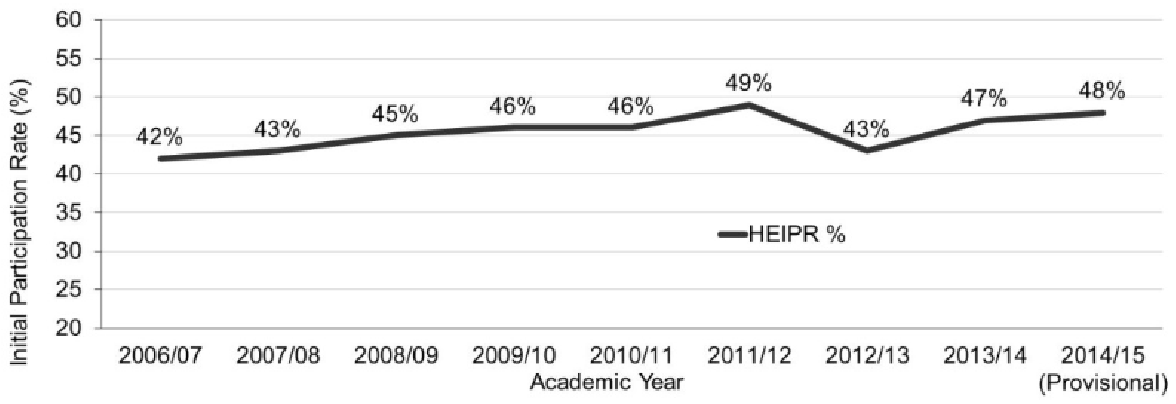


Figure 1.2. Initial University Participation Rates for School Leavers during Academic Years 2006/07 to 2014/15 (Department for Education 2016).

1.3.2 Computer science application trends

Despite the overall increase in student participation, and the growth in applications to study STEM subjects in the 2000s, computer science applications declined from 128,000 in 2004, to approximately 90,000 in the late 2000s (e-skills UK 2011; Universities UK 2015: 22). Computer science courses also tended to have a disproportionately high ratio of men to women (Carter & Jenkins 2001; e-skills UK 2011; Margolis et al. 2008; Universities UK 2015: 23; WISE 2014: 16). However, active promotion of the improved employability prospects for graduates from computer science programmes (Department for Education 2014; House of Commons Science and Technology Committee 2016) has contributed to an observed increase in applications by both men and women over the past decade. Analysis of UCAS data (Table 1.1) demonstrates a rise in the total number of computer science applications (including software engineering and games development) from 90,260 in 2010 to 129,145 in 2016.

Table 1.1. January total number of applications for UCAS Group I Computer Sciences.

Year	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016
Computer Science	53,610	47,475	53,785	59,820	64,910	60,225	68,060	76,895	87,570	90,775
Information Systems	13,965	12,515	13,930	15,925	15,855	12,425	11,350	10,800	10,165	9,305
Software Engineering	6,315	5,975	6,515	7,725	8,945	8,195	9,495	10,155	10,425	10,765
Artificial Intelligence	440	315	325	395	440	385	325	495	485	760
Health Informatics	0	0	0	0	0	30	25	20	30	25
Games	0	0	0	0	0	2,350	3,530	7,490	9,570	11,125
Computer generated Audio and Visual effects	0	0	0	0	0	45	270	425	410	575
Others in Computer Science	65	0	5	110	10	350	495	550	660	710
Combinations in Computer Science	5,325	4,845	5,320	6,285	6,940	5,930	6,150	5,470	4,905	5,105
Total applications:	**79,720**	**71,125**	**79,880**	**90,260**	**97,100**	**89,935**	**99,700**	**112,300**	**124,220**	**129,145**

Note: The source for this data is (UCAS 2016a: 3).

These applications translated into 26,850 being accepted on computer science and related courses in 2016, which represented a 33% increase from the 20,500 students accepted in 2010 (Table 1.2). Another notable trend in Table 1.2 is the sudden appearance of 'games' courses in 2012 followed by steady year-on-year growth for this emerging subject. Table 1.2 also illustrates a decline in the popularity of Information Systems. However, it is unclear as to whether this represents a real downturn in subject interest or is in fact due to rebranding of degrees that also subsume Information System subjects. These patterns demonstrate, in contrast to many 'conventional' subjects, the rapid development and emergence of new disciplines within computer sciences.

Table 1.2. August acceptances for UCAS Group I Computer Sciences.

Year	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016
Computer Science	11,420	12,135	13,255	13,245	13,405	12,565	14,390	15,755	17,730	18,260
Information Systems	3,250	3,485	3,855	4,010	3,610	2,905	2,665	2,365	2,340	2,105
Software Engineering	1,410	1,585	1,675	1,700	1,880	1,865	2,075	2,120	2,080	2,225
Artificial Intelligence	70	60	60	60	65	60	60	95	85	105
Health Informatics	0	0	0	0	0	5	5	5	5	5
Games	0	0	0	0	0	500	875	1,835	2,320	2,755
Computer generated Audio and Visual effects	0	0	0	0	0	20	80	120	125	170
Others in Computer Science	35	0	0	25	5	135	120	130	180	195
Combinations in Computer Science	1,095	1,250	1,435	1,460	1,450	1,295	1,440	1,155	1,085	1,030
Total applications:	**17,280**	**18,515**	**20,280**	**20,500**	**20,415**	**19,350**	**21,710**	**23,580**	**25,950**	**26,850**

Notes: (1) The source for this data is (UCAS 2016b: 3); (2) In the past UCAS has changed the Joint Academic Coding System (JACS) codes for classification of subjects. From 2007-2011, UCAS used JACS2, whereas, 2011-2016 used JACS3 classification. Figures from 2007-2011 are approximated from equivalent codes, but may not be exact.

1.3.3 Changing expectations of Higher Education participants in the 21st Century

As previously mentioned, tuition fees were introduced in 1998 to maintain resources and fund additional staff recruitment in response to greater student numbers. Students were initially charged £1,000 per year of tuition from 1998 to 2004 before the fee was raised to £3,000 per year in 2004 (Alley & Smith 2004). The fees increased by a few hundred pounds in subsequent years (Alley & Smith 2004) until the Conservative and Liberal Democrat coalition raised the tuition fees from £3,290 a year to £9,000 in 2012 (Coughlan 2010) following recommendations from Lord Browne's review of university funding in England (Browne 2010).

Researchers and commentators have questioned whether the sustained increase in tuition fees has led students to view higher education from a consumer perspective (Molesworth et al. 2011; Saunders 2014; Tomlinson 2014). Tomlinson (2014) set out to explore the extent to which recent financial changes in higher education had altered the way students approach learning, their expectations of their course and elected university and their prospects after graduating. Michael Tomlinson interviewed students of different ages from a wide variety of institutions. Students who participated in the survey had enrolled on courses in 2011 or 2012, and exhibited a diverse range of attitudes towards perceptions that universities were objects of consumerism. Student views ranged from active 'service users' to those who 'resisted' a consumer attitude. However, only a minority of students thought they were buying a service and were therefore entitled to a degree (Tomlinson 2014: 28). Instead, most students valued employability; seeing higher education as an investment to achieve a better starting position in the job market. Many students commented that the increase in tuition fees led them to seek courses and universities that represented 'good value for money'.

The annual 2015 UK Graduate Careers Survey corroborates Tomlinson's (2014) findings that greater tuition fees have increased expectations about employment upon graduating. The survey of 18,000 final year students from 30 research intensive universities revealed that 26% were expecting to start a full-time job after graduating (High Fliers Research 2015; Havergal 2015). This was the greatest proportion in the 14 years that the survey has been conducted. The percentage of students that started considering their career options in the first year increased from 30% in 2010 to 48% in 2015, and the percentage of those who started job searching before the third year increased from 57% in 2008 to 79% in 2015 (Havergal 2015). Half of the 2015 cohort sought work experience or internships during their time of study (Adams 2015).

The percentage of students who find employment within six months of graduating has become one of the institutional performance indicators applied to universities and courses (HEFCE 2016). Students can now compare employability ratings to find courses that represent 'good value for money'. Despite increasing numbers of vacancies in the digital and technology sectors, computing courses are reported to have some of the highest unemployment rates when compared with all other subjects (Shadbolt 2016). However, unemployed graduate figures for computer science courses vary widely between institutions, ranging from 2% to 26% (CPHC 2016: 5). Reasons for this are complex and are not always related to inadequate skills. For example, unemployment was higher in those low-tariff HEIs that did not offer a work placement year as part of a 'sandwich' degree course (Shadbolt 2016: 4). However, the wide variety of different jobs that computer science graduates apply for require vastly different skills and levels of experience. Shadbolt (2016: 5) found that employers disagreed as to which specific platforms and technologies should be taught. However, most agreed that graduates needed a strong grasp of the fundamental concepts of computer science to more easily transfer skills and knowledge between platforms and technologies used in the workplace.

Whilst promotion of lucrative career prospects and the availability of jobs has been effective in attracting computer science degree applications (which are necessary to address the shortfall in numbers of highly skilled workers), studies have also linked career impetus with strategic and performance related approaches towards learning (Elliot et al. 1999; Porter & Zingaro 2016: 281; Zingaro 2015). Porter and Zingaro (2016) found that students primarily motivated by career motivations are less inclined to concentrate on deeply understanding subject content. However, even though more students are primarily stimulated to undertake computer science courses for the perceived graduate prospects, recent surveys suggest most applicants are still attracted by their strong and fundamental interest in the subject (Department for Business Innovation and Skills 2016: 25).

1.4 Identifying reliable predictors to inform effective interventions in introductory programming courses

Sections 1.1 to 1.3 outline important themes and trends in three related areas (digital technology industry, programming education, and student participation in HE), which provide context and motivation for this author's own research. Section 1.1 described the shortfall in workers with adequate skills to meet the demand of expanding digital and tech industries. Industry leaders have criticised compulsory education curricula for failing to inspire and prepare students for undergraduate level study (Section 1.2). Furthermore, the high non-continuation rates associated with undergraduate computer science courses and introductory programming courses have also reduced the numbers of suitable graduates available for recruitment. Whilst the new computing curriculum implemented in UK primary and secondary schools in September 2014 is seen as an improvement (Computing at School & NAACE 2014), further developments to undergraduate curriculums and teaching practices are required to develop graduate skills (such as complex problem-solving) that align with industry needs (Shadbolt 2016). Section 1.3 reports that growing student participation in HE (and computer science related courses) and increases in tuition fees have coincided with changing expectations and opinions of HE amongst students. Surveys indicate that students increasingly expect higher education to prepare them for employment. As a result of such expectations, students may become 'goal orientated' towards achieving high grades and less inclined to take part in unassessed activities that do not directly contribute towards their degree classification (CPHC 2012: 23).

Widening access measures have greatly extended pathways for university entrance (Department for Business Innovation and Skills 2016: 7; New Labour Party 2001: 17; OFFA 2015). Most universities now accept alternative qualifications to A-levels. These are awarded equivalent UCAS points (UCAS 2016c: 34; UCAS 2010) and have led to increased numbers of students applying with BTEC qualifications, NVQ's or via other forms of HE access course. However, as a result of widely ranging educational backgrounds, experiences and qualifications, tutors can no longer assume that students share the same level of knowledge or experience. As a consequence, it is also more challenging to identify students who are potentially vulnerable to not continuing with their studies or to predict student progress with their learning.

Previous research comparing student characteristics and behaviours with programming performance (typically measured by assessment grades) has tended to result in moderate to weak correlations that may only have limited value for guiding educational intervention. Some of the stronger predictors have been found to be unreliable when attempts have been made to replicate findings at other institutions (Bornat 2014; Caspersen et al. 2007; Lung et al. 2008).

A recent Higher Education Academy (HEA) report examining retention and attainment in computer science courses (Gordon 2016: 18) identified specific areas for which further research and a stronger evidence base were needed, including the identification of the following:

- Reliable predictors to indicate computer science students at risk of failing or dropping out.
- How best to intervene and support students at risk, especially those who are considered to belong to minority groups in computer science courses (women and BME students).
- Examples of best practice in teaching, learning and assessment of computer science courses.
- How to address procrastination and distracted behaviour as well as supporting students to manage their time effectively to meet assessment deadlines and relieve negative stress.

The same author also made certain recommendations relating to information gaps in the literature (Gordon 2016: 19); among these were that:

- Departments should provide support during the transitional period between school and university. They should help students develop their academic skills (study skills and time management, in particular). A welfare team should also be available to respond to students' personal issues (adjusting to living away from home, financial concerns and budgeting etc.).
- Transitional support should also include activities that help develop students' social bonds and their sense of belonging to their community (Thomas 2012a).

- Academics should consider using active learning techniques such as peer learning and gamification to increase engagement and learning. Active learning techniques have also been found to improve a sense of belonging to peer groups and confidence.
- Academics should also monitor students' performance as courses progress and should investigate course data for its potential to predict performance.

Despite previously unsuccessful attempts, Gordon (2016) recommends continued investigation to provide evidence for reliable predictors and effective pedagogical interventions. Computer science courses are underpinned by "technical programming and computing architecture that provides the technical scaffolding for this digital content" (Gordon 2016: 11). Therefore, addressing retention issues associated with, and improving the teaching of introductory programming courses (Longi 2016) is paramount towards improving the quality of graduate skills, and the quantity of graduates required by an expanding industry.

1.5 The research aim and supporting objectives

There is a clear requirement to improve the learning and teaching of computer programming to meet a growing demand for graduates who are skilled in programming, problem-solving and computational thinking.

This research also extends the work of Mather (2014) and Mather and co-workers (2015) by evaluating more widely ranging elements of student background information for its potential to indicate and predict performance during the first 'level 4' (L4) year of undergraduate study. Here research is further extended to determine the consistency of patterns and findings over three years of L4 cohorts and to evaluate predictors at other institutions teaching computer-science related degrees.

1.5.1 The research aim

The core research aim may be expressed as a statement of intent as follows:

To determine predictors of success or vulnerability in learning computer programming towards supporting more timely remedial intervention and improving undergraduate progression, and to propose pedagogy for deeper learning and more effective retention.

1.5.2 Supporting objectives

It is anticipated that the following objectives and actions must be fulfilled to achieve the overall research aim. The rationale for selecting methods identified below is explained in the methodology sections of Chapter 3.

Objective 1: Review predictive trends and student behaviours already identified in the literature.

- Method: Review of existing literature concerning the difficulties students face when taking introductory programming modules (Chapter 2).

Objective 2: Investigate pedagogical interventions that have been implemented in other higher education establishments.

- Method: Review literature concerning pedagogical choices and interventions that educators may make to influence approaches to learning (Chapter 2)

Objective 3: Evaluate the usefulness of prior qualifications for preparing students to learn computer programming in a higher education establishment.

- Method: Analysis of interview transcripts, survey data, and curriculum documents to determine the extent of programming coverage and perceived usefulness of prior qualifications (Chapter 4).
- Method: Correlational analysis to determine the indicative value of UCAS points and performance comparisons between qualifications (Chapter 4).

Objective 4: Explore patterns of engagement with L4 programming learning environments towards identifying potential indicators of success and vulnerability.

- Method: Multivariate analysis of student behaviours (motivation, preferences) and course data (attendance, grade, formative test scores) (Chapter 5).
- Method: Ethnographic-type analyses of video and audio recordings to investigate the problem-solving approaches of learners and related peer communication and collaboration behaviours that may be indicative of 'successful' learning and programming strategies (Chapter 5).

Objective 5: Suggest early forms of diagnosis and pedagogic intervention useful for retaining and motivating students whose behaviours indicate vulnerability to failing or to not progressing with their studies.

- Method: Case study evaluations to determine the effectiveness of interventions and the production of guidelines to remedial pedagogy (Chapter 6).

Chapter 2 Literature Review

This chapter reviews literature surrounding the learning and teaching of computer programming and is particularly concerned with introductory level courses that are typically encountered in the first year of undergraduate level study (designated as Level 4 or L4 according to the UK Framework for Higher Education Qualifications). The first two sections explore relationships between student characteristics and their grades in introductory programming courses. The last section investigates instructional design as well as specialised interventions and pedagogies for teaching programming. This review therefore aims to establish whether reliable predictors of programming success already exist and to identify where further work is required. It also aims to evaluate interventions and pedagogies that are currently being used to improve outcomes in programming courses.

Section 2.1 reviews the impact of prior programming exposure on undergraduate programming performance and learning success. Such preceding exposure may include programming knowledge and experience acquired before university entry and/or school-level computing and IT qualifications. As well as considering the influence of ethnicity (BME grouping), gender and mature entrance to degree programmes on learning performance, this section also explores the expectations and motives of current students in further detail.

Section 2.2 investigates typical patterns of student engagement with programming courses. Learning approaches and styles, together with cognitive abilities (such as problem-solving ability and the formation of mental models) are discussed in relation to predicting programming performance. This section also discusses the impact of technology usage on attention span, engagement with taught sessions and independent study. Emotional and psychological factors such as comfort level and a sense of belonging to a peer group or wider community are explored for their influence on outcomes.

Section 2.3 considers the influence of course design and delivery style on learning outcomes. Choice of programming language, development environments, class size and supporting materials are discussed. Consideration is then given to conventional didactic teaching methods and recent pedagogies developed specifically for teaching computer programming. The advantages and disadvantages of different assessment techniques are also contrasted.

Section 2.4 reviews the findings of Sections 2.1 to 2.3 and uses these to contextualise and describe the areas that this research intends to address.

The scope and identification of relevant literature

Due to the ongoing and rapid evolution of programming languages, software development environments and teaching approaches, much of the literature relevant to this research has been published since 2000; earlier works are often no longer representative of circumstances encountered by current students (Fisler et al. 2016). However, literature from the 20th century is included where it remains relevant, particularly with regard to fundamental theories of learning and pedagogy, and where such ideas have been extended to represent contemporary practices in a digitally connected age. This chapter also underpins the development of ideas with learning and teaching research outside the specific context of teaching computer programming. Such wider consideration of approaches beyond the immediate subject domain is recommended by methodological researchers (Malmi et al. 2014).

2.1 Student preparation prior to university entrance

2.1.1 The influence of previous qualifications and education

The constructivist philosophy of knowledge creation (epistemology) emphasises the need to consider the existing knowledge that learners possess. This philosophy is widely adopted by computer science researchers and educationalists (Ben-Ari 1998; Malmi 2010). Computer science education (CSE) literature considers the effectiveness of prior education in preparing students for undergraduate study. Section 2.1.1 reviews trends in the following areas, with regard to predicting undergraduate programming performance, including UCAS points; increased uptake of BTEC qualifications; A-level and BTEC computing qualifications; previous programming experience; and mathematical ability and qualifications.

UCAS points

Universities tend to accept applications if students achieve the required UCAS points. Such points are calculated from students' A-level and BTEC grades. Despite the continued emphasis on entry requirements, CSE research has often concluded that UCAS points (entry grades) alone have a weak relationship with students' final undergraduate classification (Boyle et al. 2002; Clark & Boyle 2005; Drummond 2009). However, these investigations have revealed other patterns regarding entry grades and L4 performance.

Boyle and co-researchers (2002) investigated the extent to which A-level points of computer science students entering the University of Leeds or the University of Kent influenced L4- L6 average grades and final grade (classification). Both universities required students to have achieved A-level grades 'BBC' or above to enter their computer science courses and Leeds required students to have at least a C in mathematics (Boyle et al. 2002: 9). The researchers found that students who achieved A-level grades of 'BBB', achieved higher L4 average grades than students with A-level grades less than 'BBB' (Boyle et al. 2002: 11). However, they found that L5 and L6 grades of the two groups of students were similar (Boyle et al. 2002: 12). The authors reported great variation in student entry grades (previous qualifications) and a weak correlation between entry grade and final year grade average.

Drummond's (2009) PhD research also discovered that the relationship between entry grades and L4 grades was stronger than that between entry grades and L5 grades. Furthermore, she investigated relationships between the A-level qualifications chosen by Durham University computing students and their undergraduate grades. Students with previous maths qualifications achieved higher L4 programming grades than those without maths. However, L5 grades of students without prior maths qualifications were similar to those with maths qualifications.

Both studies indicate that the A-level Computing grades of university entrants align more closely with their L4 grades achieved at L5 and L6. Drummond (2009) discovered that A-level Mathematics grades had a similar relationship with university grades, but Boyle and colleagues (2002) saw negligible difference between undergraduate grades of those with and without A-level Mathematics. Despite this, the difference between L4 grades for the two groups is important because this is reputedly the stage at which students are most vulnerable to failing and dropping out from introductory programming courses.

<u>The increased uptake of BTEC qualifications</u>

Earlier studies (Boyle et al. 2002; Drummond 2009) only referred to small populations of non-traditional students (those who took qualifications such as BTECs, GNVQs and access courses). Drummond (2009) did not include BTEC students in her analysis of 2004-2006 Durham students because they were too few to make a valid comparison with the greater number of traditional A-level entrants. Boyle and co-workers (2002: 10) noted that for some 30% of students from both institutions (Kent and Leeds), "the background of these students ranges through non-traditional school and college qualifications (e.g. BTEC, GNVQ), to preparatory foundation or access courses geared to the mature entrant." They found negligible difference between the average grades of traditional students (A-level students) and non-traditional students for both Kent and Leeds cohorts.

Recently, BTEC qualifications have become a popular alternative qualification to A-levels. Many students choose to undertake BTEC qualifications because they are assessed primarily through coursework instead of examinations. However, Gill and Vidal Rodeiro (2014) state that the BTEC syllabus was not originally designed to prepare students for university. Instead, the qualification was intended as preparation for the workplace with assessment methods that better reflected the 'real-life' nature of the subject vocation (London Economic 2013). In contrast, the A-level was designed to be the pre-requisite entry qualification to university and assessed primarily by examinations.

Despite not being designed for university application, the number of students applying only with BTEC qualifications has increased, as has the number of those applying with a combination of BTECs and A-levels. The number of students entering university with at least one BTEC qualification has more than doubled between 2008 and 2015 from 44,000 to over 100,000 (UCAS 2016a: 27). The number of BTEC students achieving the highest 'distinction' grade is now double that of students achieving the equivalent A-grade for A-levels. HEFCE (2015) reported that 38% of students achieved three BTEC distinctions in 2012/13 (an award equivalent to 'AAA' at A-level), compared to only 17% in 2005/6. In contrast the percentage of students who achieved 'AAA' at A-level only marginally increased from 16% to 17% between 2005/6 and 2012/13.

Whilst BTEC qualifications have enabled those from disadvantaged backgrounds to access HE courses (Gicheva & Petrie 2018; Kelly 2017: 13), the qualification does not provide the same level of access to highly selective institutions as A-levels. In 2018, the Office for Students (Finlayson 2018: 1) reported that "98 per cent of students with A-level A*A*A* are going to high tariff institutions whereas only 21 per cent of students with BTEC D*D*D* are going to the same institutions." Many Russell Group universities require an A-level to be taken alongside BTEC qualifications (Russell Group 2017: 16) to provide the "broader academic skills required for degree study" (Grove 2016).

Even though the assessment methods are different, UCAS awards the same number of UCAS points for equivalent grades – a 'D' ('Distinction') at BTEC is the equivalent to an 'A' at A-level. Educators and academics have voiced concerns at this because students are being awarded the same points for quite different skill sets. Critics say that the "practical-based qualifications do not adequately prepare students for essay writing, independent study and other aspects of academic life required for degree-level study" (Grove 2016). Dalton and MacKay (2019: 10) found that progression from L4 to L5 was lower for BTEC entrants (81%) than for A-level entrants (96.5%) across a sample of 128 students studying a variety of subjects. However, the same researchers discovered that the percentage of BTEC students that progressed from L4 to L5 significantly increased if they held an A-level qualification (93%).

Trends in pre-entrance computing qualifications

A variety of computing and IT curriculums have in the past been offered and continue to be available at A and BTEC levels. ICT and Computing may be taken as separate subjects at A-level. BTEC students may elect to take IT, or specialise in Digital Games Design and Development as part of a course in Creative Media Production. The number of students choosing to study A-level Computing has increased yearly since 2014, rising from 4,171 in 2014 to 5,383 in 2015 (an increase of 29%), with a further 16% increase to 6,242 in 2016. These year-on-year increases for A-level Computing represent the greatest changes in uptake of all STEM subjects during this period (Joint Council for Qualifications 2015; Joint Council for Qualifications 2016a). However, the numbers of students studying computing was still less than that for ICT (8737) in 2016 and remain relatively small when compared with other STEM subjects including: Mathematics (92,163); Sciences (Biology: 62,650; Chemistry: 51,811; and Physics: 35,344); and Design and Technology (12,477) (Joint Council for Qualifications 2016b).

Some educators are sceptical about the value of a previous computing qualification due to a perception that the quality of award may depend on the background of the school/college and staff delivering the subject. Boyle and colleagues (2002: 12) wrote that "many students score very highly in this subject on the basis of project work that has not necessarily instilled the habits and discipline that university CS departments seek."

During the 1990s and early 2000s, both the AQA and OCR A-level Computing specifications did not require students to learn programming (Clark & Boyle 2005). This was thought to be due to the lack of programming expertise amongst A-level teachers (Clark & Boyle 2005). However, since the mid-2000s, when Clark and Boyle conducted their research, successive OCR and AQA curriculums have now incorporated programming into their specifications (AQA 2009; OCR 2009). Programming is also now being taught at all stages of compulsory education (KS1-KS4 and A-level) in the 2014 computer science curriculum (Cellan-Jones 2014; Department for Education 2013a; Department for Education 2013b). Both the 2016 BTEC Computing and IT syllabi feature programming units. These are optional for the computing course but mandatory for the diploma level IT course. It therefore appears likely that an increasing number of university entrants will possess coding skills. If learnt correctly, such skills may well be beneficial to some students who might have otherwise struggled with programming during their first year of study at university (Clark & Boyle 2005).

Although there is a significant difference between the A-level ICT and A-level Computing syllabi in relation to programming, most students fail to differentiate the two (Gordon 2016: 11). The 2013 OCR A-level ICT syllabus mentions that students are rarely expected to program (OCR 2013: 46), whereas the computing specification dedicates an entire AS unit to programming methods and techniques (OCR 2015: 13). Similarly, AQA's A-level Computing specification dedicates an AS and A2 unit to teaching programming (AQA 2014a: 5), and the ICT specification does not mention programming (AQA 2014b: 5). However, Edexcel's A-level Applied ICT curriculum offers two units on programming: one focused on creating applications (Edexcel 2013: 243) and another involving the modification of spreadsheets and databases with macros (Edexcel 2013: 205). However, programming units are optional for all Edexcel awards and their inclusion depends on which units are selected by the teacher (Edexcel 2013: 3).

Drummond (2009: 127) compared the L4 and L5 performance of A-level ICT students with A-level Computing students. She found a greater difference between the L4 programming average grade for students who took A-level Computing (at 59.47), compared with those who did not (at 53.33). However, the grade average for students without an ICT A-level (57.84) suggested that they performed better than those with the qualification, for which the average grade was 52.24 (Drummond 2009: 128). Drummond (2009: 137) also discovered that nearly 62% of 21 A-level ICT students said their qualification was unhelpful for undergraduate study, whereas only 13% of 30 A-level Computing students were of this opinion.

Whilst Drummond's study suggests that A-level Computing is more effective in terms of preparing students for undergraduate level programming, it does not account for the recent increase in students applying with BTEC qualifications. A comparison between the average grades for BTEC and A-level students has yet to be undertaken.

The impact of pre-entrance programming experience

Students may have learnt to program outside of a formal A-level or BTEC qualification through employment or independent study. Earlier research has sought to measure and compare previous programming experience (irrespective of how it was gained) with introductory programming grades. However, studies have yielded mixed results.

Hagan and Markham (2000: 27) conducted a study of 97 students during four stages of an introductory programming module at Monash University in Australia. Questionnaires were used to elicit biographical information, educational expectations and programming experience. Results indicated that students who had prior experience in a programming language performed "significantly better" in course assessment than those without experience prior to starting. The authors also discovered a positive relationship between the number of computing languages that students had prior experience of and their programming grades (Hagan & Markham 2000: 27).

In their assessment of programming experience, Wilson and Shrock (2001) asked whether students had taken a programming course before and whether they had learnt to write programs outside of a formal class. They also measured computer usage using the following indicators: internet experience in number of hours per week; number of hours per week playing games on the computer; and number of hours per week using productivity software such as word processing, spreadsheets and presentation software. A combined expression for these explanatory variables was found to be significantly correlated with the mid-term score for an introductory programming course ($r= 0.387$, $p <0.01$, $n = 48$) (Ahadi et al. 2015: 122).

Wiedenbeck and co-researchers (2004) also combined several measures of student programming expertise and computer use in a single factor. The components of the merged indicator included: the number of ICT courses taken; the number of programming courses taken; the number of programming languages students had used; the number of programs students had written; and the length of those programs. The combined indicator was weakly but significantly correlated with student score for an introductory module ($r = 0.25$; $p < 0.05$).

However, some authors report that students with little or no programming experience may perform better than those with prior experience. Bergin and Reilly (2005a) found that students with no previous programming experience achieved a greater overall mean score in an introductory programming course. In a study of 39 students Watson and co-workers (2014) also reported a weak but statistically significant and negative correlation between programming experience in years, and course points ($r = -0.20$; $p < 0.05$).

Pre-existing programming knowledge was also found to hinder learning progress in some earlier studies concerned with learning psychology (Soloway & Spohrer 1989). Other researchers found that inconsistent (Bonar & Soloway 1985) or incorrect (Lui et al. 2004) mental models of programming concepts may also hinder learning. This can be an obstacle to progress and requires an 'unlearning' process or modification of acquired knowledge (Ben-Ari 1998; Taylor & du Boulay 1987). Therefore, it appears that programming experience is not necessarily advantageous per se, but also requires programming concepts to have been learnt correctly in the first place. After all, "there is no reason to expect that previous unsuccessful attempts will be beneficial" (Robins 2010: 38).

Robins (2010: 37) maintains that predictive research would benefit from more detailed reporting of methods, results and the survey questions administered. There also appears to be little consistency in the way previous programming experience is measured. De Raadt and colleagues (2005) asked students to self-assess their previous programming experience on a scale of 1-5. In contrast, other researchers (Hagan & Markham 2000; Wiedenbeck 2004; Wilson & Shrock 2001) asked students to complete comprehensive questionnaires. In addition to determining the success of previous learning attempts, Robins (2010: 37) suggests recording how recently such attempts were made. This information can help to explain how longer periods without reinforcing key knowledge and skills may influence memory (Anderson 2005; Davidson 2011: 30).

The influence of previous mathematics experience and mathematical ability

Even though most undergraduate computer science courses do not require a prior computer science qualification, some do specify A-level mathematics for entrance (Gordon 2016: 8). Mathematics often plays a prominent role in topics taught in computer science courses, and is therefore worth considering as a predictor of programming performance. Consequently, Pears and co-workers (2008: 208) are of the view that "difficulties in mastering computer science concepts are attributed to factors such as inadequate mathematics background or lack of experience with fundamental skills such as logic or abstract reasoning".

Although some studies have demonstrated moderate and positive correlations between student mathematical abilities and introductory programming performance (Bennedsen & Caspersen 2005; Bergin & Reilly 2005a; Byrne & Lyons 2001; Pioro 2006; Wilson & Shrock 2001; White & Sivitanides, 2002), other workers have found correlations to be weak or entirely absent (Stein 2002; Watson et al. 2014; Ventura 2005). Robins' (2010: 15) review found that most studies reported a "significant but a modest correlation" between mathematical ability and programming grade.

Both Rountree and co-workers (2002) and Wilson and Shrock (2001) reported positive associations between maths and programming. However, most likely out of necessity, both studies used categorical data (was maths taken: 'Y' or 'N'; level: 'school', 'university' etc.). Had scale data been available (such as numerical grades from 0-100), this may have been more useful for quantifying the influence of mathematical knowledge. Other researchers have therefore sought to establish more precisely the extent to which mathematical ability, as indicated by the grade or mark awarded for earlier study, correlates with level of programming ability (Bergin & Reilly 2005a; Pioro 2006).

Pioro (2006) undertook a study involving 38 students majoring in Computer Science (CS) and 41 majoring in Electrical Engineering (EE). He compared students who had taken a mathematics course (Discrete Mathematics, Calculus 1 or Calculus 2) either prior to the programming course, or, taken in the same semester. The letter grades for mathematics courses were converted into equivalent point grades, then averaged before analysis with the mean scores from the programming course. He found that CS men and women who took Discrete Mathematics achieved greater average scores in programming (2.3 and 2.7 respectively) than EE men and women who took Calculus (2.0 and 2.1 respectively). In his next test he compared students who had taken Discrete Mathematics and Calculus 1 with students who had taken both Calculus 1 and Calculus 2 (but not Discrete Mathematics). Even though the mean scores for the Calculus 1/Calculus 2 combination (3.2 for men and 2.8 for women) were greater than the Discrete Mathematics/Calculus 1 combination (2.5 for men and 2.6 for women), the mean programming scores were weaker for the former group of students. The mean programming scores for CS students who took Discrete Mathematics (2.5 and 2.8) exceeded those for EE students who took Calculus courses (2.0 and 2.1). Although the EE majors had previous maths experience (two Calculus courses), Pioro (2006) attributes their weaker average scores to not having experience of Discrete Mathematics, as this is a subject in which problem-solving techniques are taught.

In a further grade-based study, Bergin and Reilly (2005a) investigated relationships between the performance of 96 students on an introductory programming course and possible determinant factors. The co-authors found, among other relationships, that the programming performance of 30 students who had taken the Irish Leaving Certificate (LC) in Mathematics was positively and significantly correlated with the grade they had achieved in LC Mathematics ($r = 0.46$ $p < 0.01$).

Stein (2002: 34) also studied relationships between grade achievements in programming, discrete and calculus mathematics. Analysis was based on 34 students who passed the first semester programming course (CS1), discrete mathematics, calculus and the second semester programming course (CS2). He found that achievement in discrete mathematics had a poor association with CS1 ($r = 0.244$) and the relationship between CS1 and calculus was even weaker ($r = 0.162$). Interestingly, discrete mathematics was found to be significantly correlated with CS2 ($r = 0.500$), but that there was virtually no correlation between calculus and CS2 ($r = 0.0363$). Student performance in CS1 was also found to be a significant and strong predictor of their CS2 performance ($r = 0.633$).

A contrasting study conducted by Watson and colleagues (2014) at Durham University found no correlation between discrete mathematics and programming course scores (r = 0.06, p > 0.05). However, this comparison was based on a small sample of 15 students. Although they found calculus was more strongly correlated with programming scores (r = 0.37), this was not significantly so (p = 0.06). The calculus correlation was based on a sample of 26 students. However, as the authors did not declare whether any of the sample had also taken discrete maths, it is not possible to make pairwise comparisons between the two maths qualifications.

Ventura (2005) found that maths SAT scores for more than 300 students were weakly correlated with success on an objects-first semester-one programming course (CS1). This study was based on a comparison of student maths SAT scores and the number of years studying maths at high school using grades for weekly laboratory exercises and programming examinations (Ventura 2005: 230).

Although many studies report positive, even if only moderately significant, correlations between mathematics ability and programming, some investigations report contrasting findings. However, the different studies have not adopted a consistent approach to measuring ability in mathematics or to interpreting ability from qualifications. Such inconsistent use of indicators is inevitable given that reported investigations cover a great breadth of circumstances and research aims. Differences in indicators and their treatments will affect the reported significance of findings, and undoubtedly also reduce the confidence in cross-study comparisons. Nevertheless, the positive relationship between problem-solving techniques (taught in discrete mathematics courses) and programming performance is of great interest from the perspective of informing teaching practice.

2.1.2 Student profile patterns

Female students in computing

The under representation of women in computer science education is extensively documented (Alvarado & Dodds 2010; Carter & Jenkins 2001; Margolis & Fisher 2002; Margolis et al. 2008). Other related STEM subjects such as mathematics also report low female participation in L3 (post-16) and L4 (undergraduate level). In 2013/14, computer science and engineering undergraduate courses had the lowest representation of female students compared with other subject groups (17% and 14% respectively) (Universities UK 2015: 23; WISE 2014: 16).

Efforts to appeal to girls and young women have resulted in a moderate increase in participation at A- or BTEC level and undergraduate level computing subjects, but much work is required to achieve a gender balance in STEM subjects overall. The total number of young women applying to computer science and related courses rose from 12,185 in 2007, to 18,490 in 2016 (UCAS 2016a), translating into 2,580 enrolments in 2007, and 3,775 enrolments in 2016 (UCAS 2016b). A-level enrolment figures reveal that more women study ICT than computing and computer science subjects (Joint Council for Qualifications 2016a). In 2016, 3,254 women took A-level ICT; considerably more than the 609 enrolling on a computing A-level (Joint Council for Qualifications 2016b).

Despite women often being assumed to be 'at-risk' because of their minority status with respect to computing studies (Gordon 2016: 18), variations in their course performance are comparable to those observed for men. Studies reveal that women achieve similar grades to men, and some outperform men on computer science courses (Cohoon & Aspray 2006; Katz et al. 2006; Roberts et al. 2011). Yet, women are also prone to withdrawing from or failing their degree courses (Margolis et al. 2008; WISE 2014: 16). Bennedsen and Caspersen (2005) similarly concluded that gender differences were insufficient to accurately predict performance in a programming course.

Reports indicate that women often feel marginalised, less confident than men, and fail to enjoy studying computer science at university (Margolis & Fisher 2002; Barker et al. 2002). Recent research found that women still perceive the computing related subjects to be male dominated (Cellan-Jones 2017), thus discouraging many women from studying these subjects.

However, pedagogical techniques that encourage collaboration amongst peers appear to be effective in improving course progression amongst women. Bagley and Chou (2007) found that women more highly rated the value of collaboration than men. McDowell and colleagues (2006: 5) also found that women who worked in pairs were more likely to complete courses and to take final examinations (88.1%) than women who worked independently (79.5%). Women who worked in pairs also reported greater confidence in their solutions than those who worked independently. Maguire and co-workers (2014) also observed that the grade improvement benefits of Pair Programming were significantly greater for women than for men.

Despite there not being any clear evidence of a gender influence in programming ability, engaging women in computer science remains problematic. Pedagogical techniques involving collaboration help improve confidence and encourage a sense of belonging amongst women. Women who persevere and establish good relationships with their peers go on to succeed in their learning (Vilner & Zur 2006). It also appears that collaborative behaviours and the level to which students develop a sense of belonging to their institutions and peer group (regardless of gender) are critical for progression (Thomas 2012b). Plymouth University regard this to be sufficiently important for student retention that they have engineered a module specifically designed to create a collaborative sense of group and university identity for individuals (Turner et al. 2017). Such sense of belonging may, therefore, also be a strong predictor for learning progression across all subject areas. Pair Programming is explored further in Section 2.3.2.

The representation of BME students

Efforts to increase participation from black and ethnic minority groups (BME) are proving successful, and this is evident from the increasing numbers of BME students enrolling each year. Computer science undergraduate courses attract a greater proportion of BME students than most other subject groups (CPHC 2012; Gordon 2016: 6), reported to be 25% of the computer science undergraduate population.

However, it is a matter of concern that BME students do not perform as well as their peers in computer science. The Council of Professors and Heads of Computing (CPHC) found that less than half of the BME student cohort received upper degrees, compared with 63% of non-BME students (CPHC 2012: 9). This trend is replicated in other subjects at undergraduate level. The Higher Education Funding Council for England (HEFCE) records that 72% of white students who achieved 'BBB' A-levels were awarded a first or an upper second classification. However, only 56% of Asian students and 53% of Black students entering with the same A-level grade profile achieved a first or an upper second (HEFCE 2014: 3).

Following an analysis of unemployment trends, CPHC (2012: 11) reported that unemployment is greater in graduates from ethnic minority backgrounds who obtained lower degree classes from 'lower tier' universities. However, they also acknowledge that most computer science students study at post-92 institutions rather than at Russell Group universities.

The Office for Fair Access (OFFA; now Office for Students) reports that mentoring schemes are one of many emerging strategies to support BME students during degree courses and into graduate employment (OFFA 2018a). However, they found that approximately 25% of higher education institutions mentioned specific strategies to support BME students in their 2017/18 access agreements. Collaborative techniques applied in programming courses have been reported to have increased confidence, enjoyment and attainment for ethnic minority students (Beck et al. 2005) as they similarly have for women (Maguire et al. 2014; McDowell et al. 2006).

Mature student trends

The number of full-time mature students (aged 21 or over) enrolling in higher education declined from 126,500 in 2009/10 to 106,700 in 2012/13 and there was a similar reduction in numbers of part-time mature students from 300,500 to 176,000 during this time (Independent Commission on Fees 2015: 17). The decision to increase tuition fees for the 2012/13 academic year is thought to be a significant contributor to the decline in part-time enrolments (Independent Commission on Fees 2015: 16). The number of full-time mature students increased marginally from 106,700 in 2012/13 to 113,700 in 2013/14. However, the number of part-time mature students further declined from 176,000 to 153,000 over the same period (Independent Commission on Fees 2015: 17).

Mature students are also less likely to complete their undergraduate course. As noted in Section 1.2, the non-continuation rate for mature students (at 17%) was greater than that for younger direct entrants (12%) for the 2012/13 academic year (HEFCE 2013: 7). One potential explanation may be that mature students often have more external demands on their time than younger students, such as work and/or family commitments (Gordon 2016: 6; OFFA & OU 2017), which affect the time available to study. Recommendations by the MillionPlus (2018) report urge HE providers to be considerate of those with such commitments.

Mature students also have less time to socialise and integrate with their younger peers or other mature students (Thomas 2015). The report by MillionPlus (2018) further recommends that institutions should provide opportunities for mature students to meet other mature students.

In response to the work and family pressures faced by mature students, certain universities offer courses taught in shorter time frames (such as one-week intensive courses, or six-week blocks) for students who are unable to devote the 20 to 30 weeks of full time commitment more typically expected in each academic year (OFFA 2018b).

2.1.3 The role of motivation

Changing motives of computing students

Section 1.3 considered the increase in computer science degree applications, and more widely, changes in student expectations of undergraduate study. It was less common during the 1970s and 1980s for students to choose their undergraduate degree strategically with their future career in mind. Students generally only elected to study computer science if they had a strong interest in the subject (Borstler et al. 2008; Forte & Guzdial 2005; Jenkins 2001a). However, there is emerging evidence that students who enrolled after the tuition fee rise are more 'career focused' (Adams 2015; Havergal 2015; High Fliers Research 2015) as many students now seek work experience, interview experience and CV development earlier in their undergraduate study. Although some students maintain a passionate interest in the subject, a growing proportion of undergraduates also perceive computing subjects to be a pathway to secure a well-paid job (Shell et al 2016: 642).

Research which aimed to determine motivations for studying computer science and programming has revealed the extent to which career desire is prevalent amongst current students. Jenkins (2001b: 55) asked 365 students (from two English universities) to record their principal motivations for reading computer science - and for taking an introductory programming course - as well as their attitude to undertaking studies. The survey highlighted that the main motivation for taking a computer science degree was 'aspiration' (40% of students selected 'career options'), but 36% primarily chose the subject to learn more about it. The most popular reason for taking an introductory programming course was because it was compulsory (48%), but surprisingly only 9% chose the course solely to further their career (without any other educational motivation). With respect to student attitudes, most expressed their wish to succeed for their own satisfaction (50%); however, 47% were motivated by their desire to gain 'good' jobs as a result. The latter is also confirmed by recent other findings that contemporary students choose to study computer science because of perceived career benefits (Lyon & Denner 2016: 14).

<u>The motivating influences of career aspiration versus subject interest</u>

Studies have demonstrated a significant correlation between 'intrinsic motivation' and programming grades. Jenkins (2001a: 54) defines intrinsic motivation as a having a deep interest in the subject being studied. In contrast, extrinsic motivation is primarily driven by career or other related rewards anticipated after completing courses. There is also a notion of 'achievement motivation' which is the desire to 'do well' for personal satisfaction (Jenkins 2001a).

Bergin and Reilly (2005b: 299) investigated relationships between student motivation for undertaking courses and their programming scores. The authors issued the motivation section from the Motivated Strategies for Learning Questionnaire (MSLQ) to 33 students. This comprised 31 questions measuring intrinsic and extrinsic motivation (among other aspects) on an ordinal Likert scale of 1-7 values. The results demonstrated that intrinsic motivation was positively correlated with programming scores ($r = 0.512$, $p < 0.01$). Students with high levels of intrinsic motivation also scored significantly higher than students who displayed medium levels of intrinsic motivation.

Other studies have shown that students who desire to understand their subject area (mastery goals) performed significantly better than those who wanted to achieve good grades (performance goals) (Nicholls 1984; Porter & Zingaro 2016: 281; Zingaro 2015). Zingaro (2015) also found that students with mastery goals reported higher levels of interest and course enjoyment than peers with performance goals. Students intending to master the subject tend to adopt the deep approach to learning more often than those focused on performance goals (Elliot et al. 1999). Surface and deep approaches to learning are explored further in Section 2.2.2.

Academics frequently express concerns that current students have different motivations for undertaking computer science degrees. Students do not always share their educator's desire to master topics and develop a deep understanding of a subject area (Haggis 2003). Some students may adopt a 'tell me what I need to know' approach in order to pass the course (Tomlinson 2014). However, being intrinsically motivated to understand topics correlates positively with programming performance (Bergin & Reilly 2015a). Section 2.2 explores learning approaches further, and Section 2.3 discusses teaching techniques that help engage students and encourage them to develop a deeper understanding of programming concepts.

Summary of Section 2.1

This section considered literature concerning student attributes and profile characteristics that may be indicative of future learning performance. Such information may be available before or shortly after enrolling on computing courses, and thereby be useful for guiding where intervention and support may be needed on programming courses.

Section 2.1.1 investigated the effectiveness of previous educational experiences in preparing students for programming courses. It considered the rising numbers of students with BTEC qualifications applying to study undergraduate courses. A-level Computing qualifications were found to better prepare students for introductory programming than A-level ICT qualifications, and it is believed that this is mainly attributable to computing qualifications including a compulsory unit that involves programming. Previous programming experience was found to be advantageous during the first year of study, but only if programming concepts had initially been learnt correctly. However, by the second year, there was little difference between programming scores of those who had previous programming experience and those who did not. Indications are that prior study of mathematics, particularly discrete mathematics, and other demonstration of mathematical ability involving problem-solving techniques were also positively associated with progress in learning programming.

Section 2.1.2 explored atypical student profiles as described by Gordon (2016: 18). Whilst participation of women on computing courses remains relatively low, there is no direct gender relationship with programming performance and the learning performance of both men and women varies greatly. In comparison to the role of gender, the profiles of BME and mature students have not been investigated as extensively. However, wider trends indicate that white peers tend to outperform BME students in undergraduate programming courses. It also appears that mature students are also more vulnerable to failing or withdrawing from courses in comparison to those who entered shortly after leaving school.

Section 2.1.3 considered whether the motivations for taking computer science courses were associated with course performance. Indications are that current students most likely undertake computer science courses for perceived career benefits, whereas those students in the 1970s and 1980s were more greatly motivated by an interest in the subject. Students driven by extrinsic motivation were found to underperform those primarily motivated by their interest in the subject. With reference to goal orientation theory (Nicholls 1984), students having mastery goals were found to perform significantly better than those motivated only by performance goals.

2.2 Student engagement in education and university life

Although it is desirable to identify students at-risk of dropping out at the earliest possible stage, the review in Section 2.1 revealed that previous experiences do not fully explain undergraduate performance. Even though some studies (Drummond 2009; Hagan & Markham 2000) indicate that previous experience gives students an advantage at L4, students of an atypical profile and without previous programming experience can also achieve high grades (Ahadi et al. 2015). Therefore, this section considers the extent to which study strategies and engagement factors impact learning performance. This section also reviews whether students succeed due to their possession of a narrow selection of programming skills or as a consequence of having general study skills, commitment and motivation to learn.

2.2.1 Attendance and its relationship with success

Research has shown that student attendance in lectures and laboratory sessions significantly correlates with their grades. This is found in both compulsory education and vocational studies. A study comparing absence and attainment in compulsory education in the United Kingdom found that sustained absence was detrimental to both KS2 and KS4 exam grades (Department for Education 2015). They found that KS4 students with no absence are 1.5 times more likely to achieve five or more GCSEs (A*-C or equivalent), and 2.8 times more likely to achieve five or more GCSEs (including maths and English) than pupils who were absent for 15% or more of their lessons over the academic year (Department for Education 2015: 5). A later replication of this study found a similar negative relationship between absence and academic performance (Department for Education 2016b).

Studies comparing attendance and attainment in higher education have shown a change in the importance of attendance. An older study by McConnell and Lamphear (1969) found little difference in undergraduate performance between students who attended regularly and those who rarely attended. One possible reason for this finding being in contrast with results from later studies is that far fewer students attended university in the 1960s, and their motivation was perhaps more greatly influenced by intrinsic interest-related factors than career aspirations. More contemporary studies do report detrimental effects for students who do not attend regularly (Colby 2004; Martinez 2001). Such variation in performance may reflect the increasing diversity of study approaches (Section 2.2.2), often associated with changing expectations and motivations towards higher education (Section 1.3 & Section 2.1.3).

In higher education, interviews with students reveal reasons that explain patterns of absence. Mature students who are parents, are often unable to attend sessions (or stay for the entire duration) that coincide with dropping their children off or collecting them from school. There has also been a rise in students taking up part time work whilst studying to meet living costs (Teague 2011: 21; Watson et al. 2014; Jenkins 2001c: 37). Watson and colleagues (2014) discovered a strong negative correlation between the number of hours a student worked part time alongside studying, and their programming performance ($r = -0.64$, $p < 0.01$). Although this correlation is based on a sample size of seven students (too small to be considered a reliable conclusion), other research based on larger sample sizes corroborates the trend that course performance decreases as the number of hours worked per week increases (Gibbs 2015: 199; Rokicka 2014; Universities UK 2005).

Even though research has shown that poor attendance is linked to poor performance, student representation bodies are often reported to be against compulsory attendance policies (Grove 2016; Feldman 2013). They argue that university students are adults and should have the right to choose whether they attend or not. Compulsory attendance (a form of extrinsic motivation) is unlikely to instil the same level of interest and passion for a subject that comes from within (intrinsic motivation). Even if attendance is not compulsory, it is often encouraged by academics for multiple reasons. Attending students can develop their social bonds and sense of belonging, as well as increasing their productivity through collaboration. Section 2.1.2 discussed the benefits of peer collaboration for ethnic minority students (and Section 2.3.2 further discusses collaborative techniques). Students who attend can also interact with their tutors and receive feedback on their work.

Although there are exceptions where non-attending students do work outside of class and achieve good grades, it cannot be assumed that all absent students are engaging with their studies. This is supported by findings from recent studies that contrast the attainment of attending and absent students (Rokicka 2014; Universities UK 2005; Watson et al 2014). The extent to which students engage with opportunities for learning is very likely to improve their learning performance. The next section explores the types of approaches students may adopt towards learning and their impact on programming performance.

2.2.2 Student learning approaches

Deep and surface approaches to learning

Ference Marton and Roger Saljo were among the earliest investigators to report that students could adopt different approaches towards learning and completing academic activities (Marton & Saljo 1976). The authors asked students to read an academic article as part of their experiment and subsequently questioned them on this article. They discovered that some students memorised information from the article just so they could answer the questions afterwards. The investigators named this the 'surface approach' to learning (Marton & Saljo 1976). Other students adopted a 'deep approach' to their learning by critically evaluating the article through thinking about structure, meaning, implications and opinions expressed and how these related to their existing knowledge and understanding. The authors observed that students who adopted a deep approach were able to answer a range of questions and remember the message of the article more effectively than 'surface' learners (Oxford Institute for the Advancement of University Learning, n.d).

This discovery prompted further research into student approaches to learning. The findings of such research have begun to change how courses are structured and taught at universities (Matheison 2015). Course materials, assessments and teaching content are now better designed to suit the needs of the student, in a form that is termed 'student-centred' learning. In their evaluation of an approach to support 'student-centred' learning, Biggs and Tang (2011: 23) found that it was the "learner's perspective [that] determines what is learned, not necessarily what the teacher intends should be learned." This specific example was underpinned by Marton and Booth's (1997) research into student learning approaches as well as by constructivist beliefs towards knowledge creation (Ben-Ari 1998). Many educational researchers (Biggs 1987; Nicholls 1984; Porter & Zingaro 2016; Zingaro 2015) have also observed that a student's study approach is strongly associated with their motivation to study.

This shift in focus also coincided with the emerging 'constructivist' learning theory, which believes that learners 'construct' knowledge by critically thinking about what they have heard or read, and reconciling this with their previous experiences and knowledge. Many consider the deep approach to learning to be an application of the 'constructivist' learning theory. This view has led to activities such as discussion and reflection being included in teaching sessions and assessments in many subject areas. Computing academics and computer science educational (CSEd) researchers have adopted this view (Ben-Ari 1998; Malmi et al. 2014).

The value of deep learning as a predictor of performance

De Raadt and colleagues (2005) conducted a multi-institutional study with universities from Australia, New Zealand and the United Kingdom to explore the relationship between approaches to learning and performance in programming courses. They used the revised two-factor study process questionnaire (Biggs et al. 2001), which produces separate scores for deep and surface learning approaches. The questionnaire featured 20 questions requiring students to self-assess their preferences using a 5-point Likert scale. A total of five 'DM' (deep motive) and five 'DS' (deep strategy) questions are used to calculate an overall 'DA' (deep approach) score. Similarly, the sum of outputs from five 'SM' (shallow motive) and five 'SS' (shallow strategy) questions form the 'SA' (surface approach) score. The maximum scores for each of the two approaches (deep and surface) is 50. The authors determined scores greater than 30 are more strongly indicative of one approach being dominant over the other and, conversely, that scores lower than 30 to indicate the weak influence of a particular approach.

An analysis conducted by de Raadt and co-researchers (2005: 7) discovered a positive relationship between DA scores and students' grades for seven out of nine institutions (r values were all greater than +0.3 with the highest at +0.63). Likewise, seven out of nine SA scores were negatively correlated with student marks (all r values were more greatly negative than -0.2 with three approximately at -0.5). However, only a small number of students participated from most of the institutions invited to the study. Most institutions (8 out of 9) provided between seven and 14 participants; the main exception being one institution with 39 participants. The authors also compared individual question responses with programming grades. Question 6 ('*I find most new topics interesting and often spend extra time trying to obtain more information about them'*), and question 13 ('*I work hard at my studies because I find the material interesting'*) were positively correlated ($r = 0.31$; $p = 0.001$) with student programming grade.

Richardson and colleagues (2012: 367-368) conducted meta-analysis with over 7,000 students from multiple institutions looking for student predictors of Grand Point Average (GPA) scores (non-subject specific) between 1997 and 2010. He noticed that the surface learning approach weakly and negatively correlated with academic performance (r values ranged from -0.1 to -0.3; average: $r = -0.18$; $p < 0.05$) whereas the deep learning approach had a slightly positive influence on learning (r values ranged from +0.1 to +0.2; average: $r = 0.14$; $p < 0.05$). However, a strategic approach to learning (defined as the application of both surface and deep approaches) was slightly more strongly and positively correlated with academic performance than for those adopting a deep approach only (r values ranging from +0.175 to +0.3; average: $r = 0.23$; $p < 0.05$).

Other authors also noticed that the strategic combination of both deep and surface approaches can result in higher grades (Matheison 2015; Zingaro & Porter 2016). However, a strategic approach to assessment does not necessarily mean that programming concepts are understood deeply enough to be used elsewhere. When studying first year programmers, Mather (2015: 13) observed that some students tended to adopt "surface, strategic or achievement approaches." These placed greater emphasis on completing tasks rather than applying a "deeper reflective understanding" necessary for more fully grasping underlying programming principles and structures. Likewise, he observed that some students were proficient in solving exercises in one programming environment but struggled to solve questions outside of that environment. A later study by Mather (2014) found that students can achieve high grades in programming courses through strategic approaches yet are unable to answer basic knowledge questions about programming concepts.

Research has shown that consistently adopting the deep approach to learning correlates more strongly with programming performance than the surface approach. Even though researchers observed learners to adopt an approach to learning consistently in their studies, pressures and constraints can cause learners to change their approach throughout the course (Coffield et al. 2004a; Matheison 2015). Simon and colleagues (2006: 193) commented on the work of de Raadt and co-researchers (2005), with the observation that "these correlations, while significant, were only indications of a trend rather than strongly suggesting that all students who adopt a deep learning approach are likely to succeed and those who employ a surface approach are likely to fail." Adopting the deep learning approach does not guarantee higher exam grades (Roddan 2002: 29). However, given how connected computer science concepts are (Robins 2010), a deep investigative approach that emphasises the underlying patterns and links between concepts would be advantageous not only in learning programming concepts, but also passing exams (Zingaro & Porter 2016: 283).

Learning Styles

A learning style differs from a learning approach. Students may choose between a deep approach in order to understand, or a surface approach intended to pass assessments (Zingaro & Porter 2016), but a learning style is a preference for learning through a specific instructional technique. Popular forms are aural instruction, utilising multimedia and visual demonstration, demonstration that involves movement and active participation as well as conventional reading and writing activities.

Models and questionnaires were designed throughout the 1980s and 1990s to determine students' preferred instructional technique. Some of the most notable models are: Kolb's Learning Style Inventory (LSI) (Kolb 1984); the Felder-Silverman Learning Style Model (FSLSM) (Felder & Silverman 1988); the Learning Styles Questionnaire (LSQ) (Honey & Mumford 1992); and the VARK questionnaire (acronym for Visual, Aural, Read/Write, Kinesthetic learning preferences) (Fleming, 1995). Other questionnaires also consider personality, such as the Myers-Briggs Type Indicator (Myers & McCaulley 1985). The Index of Learning Styles (ILS) compiled by Felder and Solomon (1997), combines the categories proposed by Kolb and Myers-Briggs, namely creating: 'active/reflective', 'sensing/intuitive', 'visual/verbal' and 'sequential/global' learning types.

By identifying learning styles educators are able to tailor teaching materials to accommodate individual preferences. However, the widespread adoption of learning style questionnaires has been criticised by many educationalists and researchers. Coffield and colleagues (2004b: 57) argue that little and inconsistent theoretical grounds exist for many of the popular models, often due to an absence of objective measurement of preferences. Other authors found little evidence that utilisation of learning style assessments has significantly improved achievement or motivation (Coffield et al 2004b; Pashler et al. 2009: 105).

Patterns amongst learning style research

Despite criticism regarding the effectiveness of learning styles adoption, CS educators sought to explore patterns amongst the learning style preferences of CS students. Some researchers have attempted to identify these preferences for CS and programming students whereas other researchers have evaluated the relationship between preferences and grades to determine whether styles have any predictive value.

Leading models for learning styles and their relevance to computer science are discussed below.

Kolb's learning styles

Concepts taught in computer science tend to be abstract by nature (no physical representation), and researchers have found that students are inclined to gravitate towards the 'converger' and 'assimilator' or abstract pole of Kolb's model (Byrne & Lyons 2001; Wu et al. 1998) and away from the pole representing 'concrete experience' (Figure 2.1). Hudak and Anderson (1990) distributed Kolb's learning style inventory questionnaire (Kolb 1985) to 94 students on a computer science course and a statistics course. They found that the "absence of a reliance" on the "concrete experiences" learning style predicted success in both introductory statistics and computer science courses. Therefore, students who are able to conceptualise and form mental models of abstract concepts are likely to have a stronger grasp of concepts compared with those who prefer to work with physical representations of ideas and concepts.

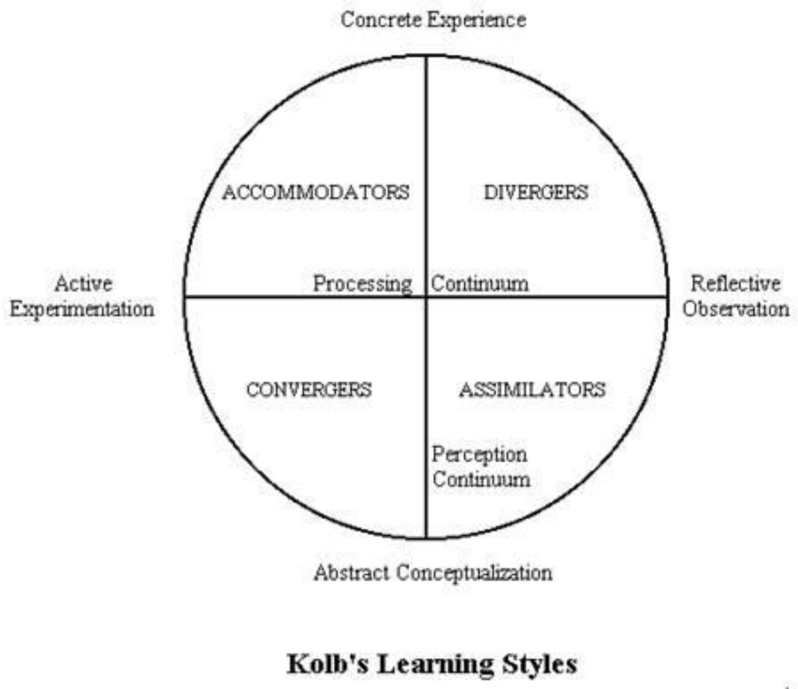


Figure 2.1. Kolb's Learning Styles Diagram (Kolb 1985).

Myers Briggs Type Indicator

Woszczynski and colleagues (2004) measured learning styles using Krause's (2000) framework of the Myers-Briggs Type Indicator questionnaire. They discovered that learners who described themselves as 'intuitive thinkers' performed better than 'sensory feelers' (Figure 2.2). The researchers found "no other differences in performance between other paired profiles" (Woszczynski et al. 2004: 1).

Pocius (1991) similarly found that programmers considered themselves to be 'intuitive thinkers' instead of 'sensory feelers', but any differences in such self-perception were not indicative of programming grades. Whitley (1996) thought that Pocius' conclusions were not representative of the wider population because Pocius had only distributed questionnaires to a small sample size of students. Whitley collected data from 471 introductory psychology students. Whitley (1996: 400) found a modest relationship between the intuitive preference and computer aptitude test results. However, he found no correlation between programming performance and the Myers-Briggs preference types of Extraversion and Introversion or between Thinking and Feeling types.

Pocius (1991) discovered that most programming students were categorised as having an introverted personality, but introversion was not related to programming grade achievements. Whitley (1996: 400) believed that this was because "introverted people may be drawn more to the work conditions of programming than to the programming task". However, the extent to which a student considers him or herself to be an introvert may not necessarily be indicative of any inherent ability to participate in peer learning activities or socialise with their peers (Granneman 2018).

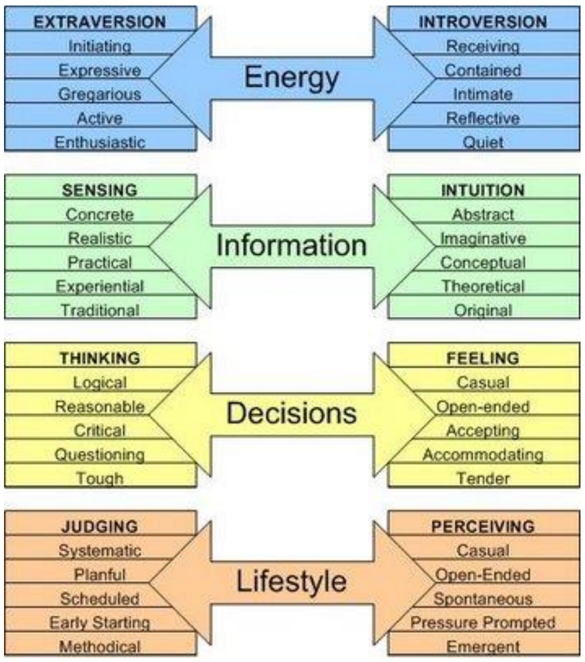
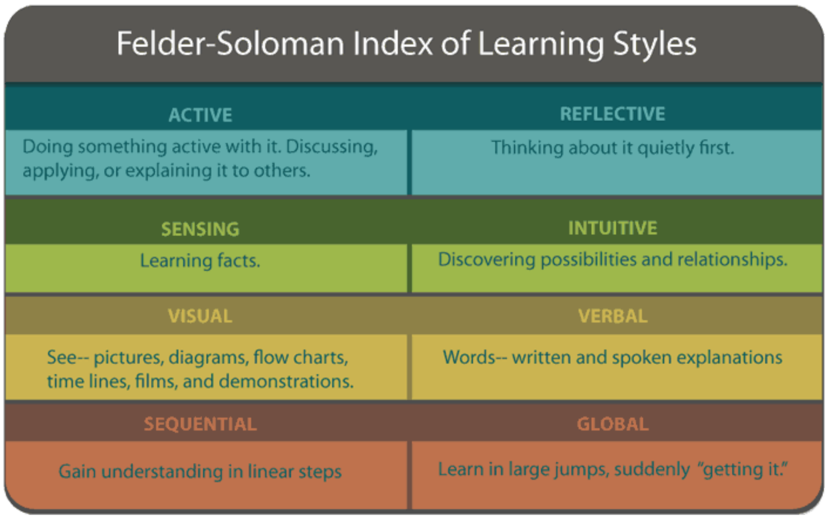


Figure 2.2. Myers-Briggs Type Indicator categories (Myers & McCaulley 1985).

The Index of Learning Styles

Allert's (2004) study determined the learning styles of more than 200 students using the ILS questionnaire. Analysis revealed that those who preferred reflective and verbal learning styles tended to perform better than active and visual learners respectively (scale opposite categories in Figure 2.3). Lewis and colleagues (2005) also discovered that preference to learn 'visually' did not correlate with programming performance. However, Rahman and du Boulay (2013) found no significant difference between the performance of active and reflective learners in a test of programming ability. Similarly, Dehnadi (2009) did not find any of Felder and Silverman's psychometric attributes to influence ability to learn to program, neither in his own research nor in work that he reviewed. For that reason, he concluded that it was an "unproductive area of research" (Dehnadi 2009: 58).



Felder-Soloman Index of Learning Styles

ACTIVE	REFLECTIVE
Doing something active with it. Discussing, applying, or explaining it to others.	Thinking about it quietly first.
SENSING	**INTUITIVE**
Learning facts.	Discovering possibilities and relationships.
VISUAL	**VERBAL**
See-- pictures, diagrams, flow charts, time lines, films, and demonstrations.	Words-- written and spoken explanations
SEQUENTIAL	**GLOBAL**
Gain understanding in linear steps	Learn in large jumps, suddenly "getting it."

Figure 2.3. Fender and Solomon's Index of Learning Styles (Felder & Solomon 1997).

VARK learning styles

In their analysis of 226 completed VARK learning style questionnaires, de Raadt and Simon (2011: 111) did not find a single dominant learning style. Nevertheless, they did discover that most students (67%) had multiple preferences in the VARK classifications, many of which included some combination with a kinaesthetic (learning by 'doing') style. The authors also found that most students expressed a preference for the read/write learning style (73%). Other researchers believe that visual learners are at a disadvantage. Milkova and Petranek (2015) observe that information is seldom presented through visual means in many institutions and that most information is still disseminated via spoken lectures (verbal learning preference) or read in textbooks.

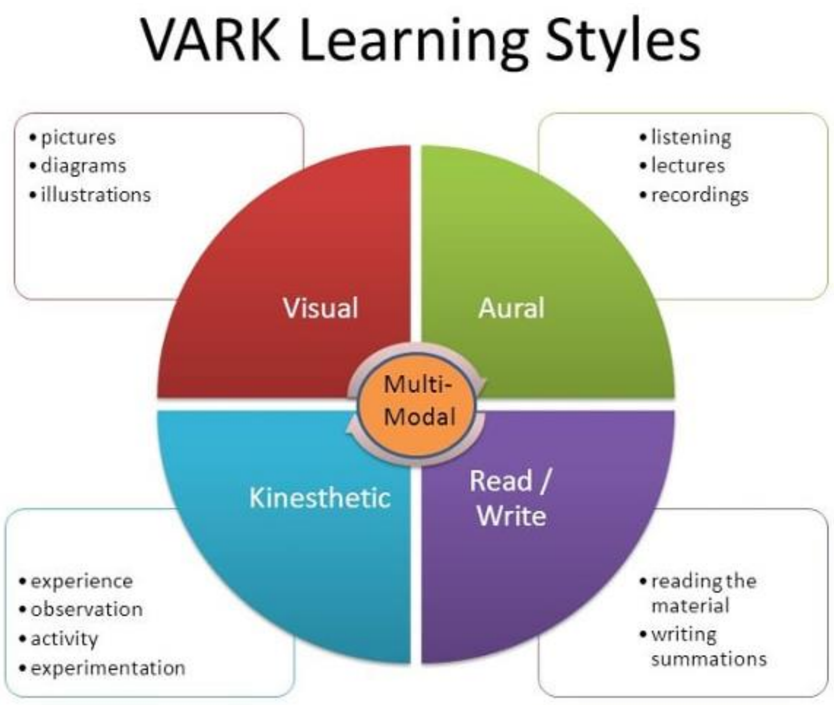


Figure 2.4. VARK Learning Styles (Fleming 1995).

Bednarik and Franti (2004) found that students who possessed similar learning preferences to their instructors performed better than those who did not. However, other studies have found a lack of alignment between instructors' and students' preferences (Bednarik & Franti 2004; de Raadt & Simon 2011). Whilst de Raadt and Simon (2011: 109) did not correlate preferences with programming grades, they discovered that instructors had a dominant kinaesthetic preference for learning (in all likelihood this was related to teaching preference), but only 16% of students shared this exclusive preference. De Raadt and Simon (2011: 111) suggest that "if learning materials are designed with good textual material to suit the read/write modality and good hands-on experience to suit the kinaesthetic, such materials should suit the learning preferences of virtually all of the students".

Summary of learning styles research

Studies utilising the Kolb and Myers-Briggs models indicate that students lean towards logical thinking and abstract preferences (Byrne & Lyons 2001; Hudak & Anderson 1990; Wu et al. 1998). Students with these preferences appear to outperform those with opposite preferences (sensory, feeling, and concrete experiences) (Haduk & Anderson 1990). Other research investigating personality indicates that programming and computer science students tend to be more introverted than extroverted, but this is not a reliable predictor of programming grades (Pocius 1991; Whitley 1996). Studies investigating Index of Learning Styles and VARK categories found greater variation in results (Allert 2004; de Raadt & Simon 2011; Lewis et al. 2005), with some findings indicating that most students preferred the read/write and kinaesthetic styles, but often in combination with other delivery modes (de Raadt & Simon 2011). Most teachers had an exclusive kinaesthetic learning preference (de Raadt & Simon 2011). Bednarik and Franti (2004) discovered that students who have similar preferences to their teachers tend to outperform those with different preferences.

Despite finding common tendencies and preferences of programmers and CS students, these have limited value as predictors of programming grades (due to contrary results). Contradictory results may stem from the research design, e.g. due to measuring variables differently and studying them under different conditions (Woszczynski et al. 2004). In addition, attempts to measure the predictive ability of one variable in isolation may be too reductionist if patterns are dependent upon a combination of factors. However, weak correlations may also indicate that specific instructional preferences have minimal influence on programming grade, if indeed students are comfortable with multiple teaching styles.

It is important to note, however, that there is key difference between the assumptions that underpin the learning style and learning approach schools of thought. Both Kolb (1985) and Dunn (1990), whose work inspired the contemporary learning styles questionnaires and tests, believed that learner's cognitive styles are determined by their psychological attributes, and therefore, unlikely to change in the short term. Sadler-Smith and co-researchers (2000) emphasise that a person's cognitive style is fixed, having been acquired and embedded from a young age. Whereas, the pioneers of learning approach research (Entwistle & Ramsden 1983; Marton & Saljo 1976) noticed that learners can freely choose which approach to adopt, regardless of previous experiences and background. This view is still maintained today by learning approach proponents such as Biggs and Tang (2011: 28) who observe that students possess the ability to strategically manage their learning approach, and choose either surface or deep approaches when needed.

Problem-solving ability

Even though problem-solving ability is commonly mentioned as an attribute of expert programmers, attempts to evaluate its relationship with success in programming courses are few (Lishinski et al. 2016). Problem solving ability is difficult to measure accurately and it is difficult for novice programmers to identify defined ways of solving a problem. Novice programmers do not possess a range of multiple problem-solving strategies (Gibbs 2015: 198) and few plan their solution before coding (Brown 1992).

These problem-solving and computational thinking skills are rarely communicated explicitly by conventional teaching materials (Bennedsen & Caspersen 2005; Kolling & Barnes 2008; Livingstone & Saaed 2017). As a consequence, most students struggle to know how to start solving a problem or find themselves occupied by syntax errors and focused on solving the task (Brown 1992).

Lishinski and co-researchers (2016: 330) noticed that previous research focused on the "potential causal relationship between the instruction in programming and generic problem-solving ability" instead of its predictive ability. Palumbo (1990) conducted a meta-review of 1980s literature searching for a causal connection between problem-solving and programming but found little evidence to support the idea. Mayer and colleagues (1986) investigated whether problem solving skills could predict programming performance and found meaningful results. Whilst positive and significant relationships existed between problem solving and exam scores, the problem-solving ability of students was measured by their ability to solve 'math word problem translation tasks' (Mayer et al. 1986: 606). Lishinski and colleagues (2016) think this is too specific to support the utilisation of 'general problem-solving ability' for predicting programming performance.

Lishinski and co-researchers' (2016) own work used problem-solving items from the Programme for International Student Assessment (PISA). Whilst this framework has been previously applied in programming research, it is also reported to be a valid measuring instrument in other fields (Lishinski et al. 2016). The authors used four scenarios from the ten item 2003 PISA test and each item was accompanied by one to three questions on a computer science topic (PISA, 2003). The authors chose four tasks (two system analysis and design tasks, one decision making task and one troubleshooting scenario) to measure problem-solving ability. They found problem solving ability to be moderately predictive for students' project marks ($r = 0.41$, $p = 0.152$), but not for exam grades ($r = -0.01$, $p = 0.517$). There was a positive correlation between exam and project scores ($r = 0.43$, $p = 0.011$), suggesting that problem solving ability "accounts for performance on programming projects above and beyond exam scores" (Lishinski et al. 2016: 332).

Even though Lishinki and co-researchers (2016) report problem-solving ability to be predictive of programming performance, this research is limited by being unable to explain the underlying cognition which enables learning progress (Mayer et al. 1986: 610). Studies indicate that the specific cognitive skills that most strongly correlate with programming performance are abstract reasoning and the ability to construct accurate mental models (Barker & Unger 1983; Dehnadi 2009; Gibbs 2000; Kurtz 1980; White & Sivitanides 2002).

Educators (Carver 1986; de Raadt 2008; Klahr & Carver 1988; Palumbo 1990) discovered that the clear and systematic teaching of planning for developing programs, and other metacognitive strategies, improved student problem-solving ability. Carver (1986) taught problem-solving skills explicitly through a five-phase model (program evaluation, bug identification, program representation, bug location, and bug correction). She found the prioritisation of problem-solving skills to be more effective than conventional modes of instruction which emphasise programming concepts and expect students to deduce problem-solving skills for themselves.

<u>Mental models</u>

Cognitive research shows that students create internal models of intangible concepts in order to relate to them. Students will successfully learn programming concepts when they create a mental model that resembles the actual concept (the 'design model') (Lui et al. 2004). Studies have shown that students who create accurate mental models of programming concepts (especially variables and assignment) and consistently apply them, outperform those who do not (Bornat 2014; Dehnadi 2009; Ma et al. 2009; Lui et al. 2004). This corroborates the findings from learning styles research and indicates that students comfortable with 'abstract conceptualisation' perform better than those who require 'concrete experiences' (Hudak & Anderson 1990).

Dehnadi (2009) discovered that students (regardless of institution) with consistent mental models had an 85% pass rate, but students with inconsistent mental models had a 36% pass rate. He defined consistency as adopting the same mental model for all or most questions that utilise a programming concept. Thus, inconsistency is applying different mental models for the same programming concept in different questions.

Many have investigated mental model consistency (MMC) with respect to the concept of programming variables, finding much variation in student understanding and the effectiveness of mental models in predicting programming performance (Dehnadi & Bornat 2006; Caspersen et al. 2007; Ford & Wenema 2010; Lung et al. 2008; Radermacher et al. 2012). Variables and the assignment of values to variables are typically covered early in introductory programming courses, yet many students continue to hold inconsistent and inaccurate models weeks later in their courses (Bornat et al. 2008). Whilst several MMC tests exist for variables and assignment, the relationship between MMC and other foundational concepts are still to be investigated in the same depth (e.g. programming structures for iteration or repeating actions) (Radermacher et al. 2012).

Consistent and correct understanding of basic concepts is essential to enable students to understand concepts taught later in the introductory programming courses (Robins 2010). The development of accurate mental models is also thought to be hindered by unsuitable teaching methods (Margulieux et al. 2012). The related issue of cognitive overload is explored further in Section 2.3.2 as well as the influence of certain types of instruction on improving the creation of accurate and consistent mental models.

2.2.3 Digital technology usage in the learning environment

<u>Media multitasking and procrastination</u>

The current generation of university students routinely utilise digital technology for work and social activities (Madden & Zickuhr 2011; Ofcom 2014). Students regularly conduct activities on their devices whilst listening to a lecture or reading (Hammer et al. 2010; Rideout et al. 2010). However, research also indicates that such tendencies can lead to distraction and procrastination, thereby delaying and undermining assessment work (Gordon 2016: 12; Smith 2010).

Studies in the early 2000s found that students using laptops for unrelated activities performed worse in memory related tests and assignments than peers who did not (Fried 2008; Hembrooke & Gay 2003; Kraushaar & Novak 2006). Kraushaar and Novak (2006) recorded that students opened an average of 65 laptop windows (internet windows or tabs) per lecture, most of which (40 on average) were unrelated to lecture content. Their study also found that students attempt to multitask 42% of the time they are in class. Sana and colleagues (2013) noticed that students sitting in view of peers attempting to conduct multiple activities on their laptop in lectures were distracted by the person attempting to multitask. In a controlled experiment the researchers found that students seated near multitaskers performed worse on a test based on the lecture content compared with students who were sitting next to non-multitasking peers (Sana et al. 2013). Kirscher and Karpinski (2010) discovered that students who regularly visited social media websites such as Facebook during class were outperformed by those who did not regularly visit such websites.

The frequent sending and receiving of text messages during lectures also have a detrimental effect on student grades. Rosen and colleagues (2011) allocated 185 participants into three groups: no/low texting interruption (0-7 text messages sent and received); moderate texting interruption (8-15 texts); and high texting interruption (16 or more texts). The groups were then texted whilst watching a recorded 30-minute lecture (which was usual practice for their course) and were asked to answer questions regarding the information conveyed immediately afterwards. The researchers found that the most frequently interrupted group of students (sending/receiving 16 or more texts in total) scored a whole grade lower (10.6%) than both the low and moderate groups in the test afterwards. Interestingly, there was no significant difference in scores between the low and moderate groups. Even when text messages were received at crucial points during the lecture, infrequent texts did not appear to hinder student performance.

In a similar study, Barks and colleagues (2011) divided their smaller number of participating students (n = 37: 7 men, 30 women) into two groups: one group which would receive and reply to text messages and the other group who would not. The researchers sent 15 texts to the texting group during the lecture, asking them to respond quickly. The group which received and sent texts were outperformed in a subsequent test by the non-texting group.

Whilst texting and surfing the web was found to negatively impact on grades, listening to music alongside studying had minimal effect (Rosen et al. 2011). Kirschner and Karpinski (2010) believe this is due to the level of attention given to activities being conducted alongside the primary activity. Texting and surfing the web require active participation which divert attention away from learning activities. However, music can be played in the background without becoming the primary focus.

Leinikka and co-researchers (2014) from Finland investigated the specific effect of task switching on learning to program. They found no correlation between task-switching and programming exam results or first-semester average grades. However, they did find a positive relationship between the number of tasks that students attempted to conduct simultaneously, and the time taken to resolve programming errors.

2.2.4 Emotional wellbeing and psychological factors

Research has shown that academic ability is significantly affected by wellbeing and personal circumstances. Students who experience distress and discomfort are less likely to succeed (Brooks & DuBois 1995; Leafgran 1989; Szulecka et al. 1987; Richardson et al. 2012; Van Heyningen 1997). Indeed, some of the non-continuation rate (previously discussed in Section 1.2) is attributable to students withdrawing for personal reasons and being unfit to study.

Although learning is primarily a result of cognitive processes, thoughts unrelated to learning contribute to cognitive load. Fear, anxiety, stress and frustration often hinder learning and cognitive performance (Rogerson & Scott 2010; McQuiggan et al. 2007).

Comfort level

Computer science researchers, in their search for predictors of success in programming, have included variables such as self-esteem, comfort level (how comfortable one feels in an environment), fear and anxiety, and emotional wellbeing into their studies (Bennedsen & Caspersen 2008; Rogerson & Scott 2010; Wilson & Shrock 2001).

Wilson and Shrock (2001) issued Ramalingam and Wiedenbeck's (1998) Computer Programming Self-Efficacy test, and their own questionnaire, to 105 students after their midterm exam. The two testing mediums measured 12 possible predictors: mathematics background, attribution of success and failure (luck, effort, difficulty of task, and ability), domain-specific self-efficacy, encouragement, comfort level in the course, work-style preference, previous programming experience, previous non-programming but computer-experience and gender. The authors discovered that comfort level on the course was the strongest predictor of success. Ventura (2005) also discovered that student effort and comfort level were the strongest predictor of success in a computing degree.

Bennedsen and Caspersen (2008: 1) asked 100 CS1 students to self-report their attitude towards perfectionism, self-esteem, coping and perseverance, affective states and optimism. They found no correlation between grades and the self-reported emotional health and social well-being metrics. However, they noted that students who pass have "statistically significant higher self-esteem than those who do not". Bergin and Reilly (2005a) measured 15 factors in a series of studies (including prior academic experience, comfort level, self-perception of programming performance and prior computer experience) on a first-year object-oriented course. Bergin and Reilly (2005b: 413) also found strong evidence that comfort level (post-course) was correlated with success ($r = 0.55$, $p < 0.01$).

Summary of Section 2.2

This section evaluated literature regarding learning metrics which cannot be deduced from prior qualifications and entry applications. It considers the influence that characteristics such as student preferences and approaches to learning; attendance; and engagement with teaching materials, have on programming performance. This section also establishes links between the aforementioned attributes, and those discussed in Section 2.1 (preparatory qualifications, student profile attributes and motives).

Section 2.2.1 found that attendance of teaching sessions correlates moderately strongly with programming grade and that sustained absence has a detrimental effect on student performance. Students are increasingly undertaking part-time jobs to help fund their living expenses. Working more than 10 hours a week alongside studying was found to have a detrimental effect on course performance (Watson et al. 2014). Both young and mature students who have other responsibilities and commitments may be disadvantaged by the conventional weekly and linear teaching format. Alternative teaching arrangements are discussed in Section 2.1.2 and in Chapter 6.

Section 2.2.2 discovered that the deep learning approach had a stronger correlation with programming performance than the surface learning approach. Section 2.1.3 found that intrinsically motivated students tend to have mastery goals, and Section 2.2.2 discovered that these motivations tend to align with adoption of a deep learning approach (de Raadt et al. 2005; Zingaro & Porter 2016). Although learning styles were generally poor predictors of programming success, they did identify common personality traits and a leaning towards abstract and logical thinking preferences. This aligns with mental model research; those who form accurate mental models of programming concepts generally outperform those with incorrect and inconsistent models (Bornat et al. 2008). However, the creation of mental models is hindered by inappropriate teaching methods (Margulieux et al. 2012) and materials which rarely explicitly teach problem-solving skills (Bennedsen & Caspersen 2007). Section 2.1.1 discovered that discrete mathematics courses explicitly taught problem-solving strategies, and students who took this course outperformed those who did not in programming assessments (Pioro 2006). Teaching methods are explored further in Section 2.3.

Section 2.2.3 established that excessive procrastination and media multitasking during teaching sessions and independent study detrimentally affected learning performance. However, low or moderate distraction had less impact on grades than frequent distraction (Rosen et al. 2011). Task switching did not align with grades but the extent of multi-tasking did align with the time taken to develop solutions (Leinikka et al. 2014). Despite the negative effects of over-using digital technology, there are also innovative opportunities to use current technology trends for facilitating learning (Davidson 2011: 64).

Section 2.2.4 found that emotional distress and low emotional wellbeing are common factors for students withdrawing from courses. Stressful and uncomfortable situations can be a source of distraction, directing concentration away from studying (Bergin & Reilly 2005b). Conversely, high levels of comfort and self-esteem were indicative of learning success (Wilson & Shrock 2001). Tutors do not have control over life circumstances pertaining to students, but attention to course design can reduce stress related to the academic context. Course design is discussed in Section 2.3.

2.3 Programming Pedagogy

2.3.1 The components of programming courses

Conventional format for teaching programming

Most introductory programming courses are taught by a format of weekly lectures accompanied by laboratory sessions (Barker et al. 2002; Kolling & Barnes 2008; Kolling & Barnes 2004). Programming concepts and language constructs are typically presented in the lecture (Kolling & Barnes 2004: 1) and the students then work on related exercises and assignments during accompanying laboratory sessions.

Whilst most computer science departments continue to deliver introductory courses in this format, there are other components of course content, supporting materials, teaching style and assessment which do vary from institution to institution.

Class size

Lectures are typically delivered to all students at once for larger classes (typically 100 students or more). Some institutions do divide the cohort into smaller classes and assign tutors to teach each group and supervise their laboratory sessions. Interestingly, Bennedsen and Caspersen (2007: 34) discovered that small classes (less than 30 students) tended to have a higher pass rate than larger classes. The average pass rate for smaller classes (representing 23% of the total number of classes surveyed) was found to be 82% whereas large classes (77%) had an average pass rate of 69%.

Curriculum design: order concepts are taught

Introductory programming course curriculums have evolved considerably over time. The recent focus on employability has increased communication between universities and industry to inform course content, programming styles, languages and popular programming techniques (Shadbolt 2016). Nevertheless, there is a lack of consensus amongst employers and academics regarding which topics should be included in the curriculum (Hertz 2010; Hertz & Ford 2013; Shadbolt 2016: 5). Generally, instructors make the final decision and tend to allocate time on each topic based on student performance (Hertz & Ford 2013).

The order in which concepts are taught is commonly discussed in literature and particularly the placement of object-oriented concepts. Traditionally, procedural and functional curriculums would adopt a bottom-up approach (Meyer 2003; Pedroni & Mayer 2006) for teaching concepts such as variables and control structures first before data structures, recursion and functions. However, some educators (Kolling & Barnes 2004) started teaching object-oriented concepts first (top-down approach), because the object paradigm maps closely to real entities in industry and the workplace.

Despite an ongoing debate, Bennedsen and Caspersen (2007: 33) noticed that there was little variation between the student pass rate for objects-first courses and imperative-first courses, amongst those that they surveyed.

<u>Programming language</u>

Currently, Java, C family languages (C, C#, C++) and Python are the most commonly taught languages on introductory programming courses in the UK (Murphy et al. 2016: 3) and internationally (Guo 2014). These specified languages are also the five most commonly used by globally leading organisations to develop applications for their business and other services (Bouwkamp 2016).

Python recently overtook Java as the most commonly taught language on the computer science courses of the top 39 American universities (Guo 2014). Ateeq and colleagues (2014) preferred Python to C++ because the syntax is closer to the English language (pseudocode) and the development environments that accompany it are more accessible. Less time spent on navigating syntax errors also means that students can devote their time to problem-solving and algorithmic thinking (Yadin 2011).

Endeavours to reduce the confusion caused by learning syntax have led to the development of 'drag-and-drop' programming languages (also known as 'code blocks') - which involve dragging and ordering pre-written code from menus (Hundhausen et al. 2009). However, the same criticisms made of older 'learner-friendly' languages are applied to these recent innovations. Jenkins (2001a: 43) refers to earlier discussions about languages developed mainly for teaching, such as BASIC, LOGO and Pascal, and asserts that few universities "would seriously consider using them due to their lack of industrial application" (ACM 2013). It should be noted that, whilst industry standard languages and environments are desired for increasing employability, their complexity can be overwhelming for novice programmers. Despite this tension, industry currently emphasises the importance of generic skills (such as computational thinking and complex problem-solving skills) and is less concerned with a person's experience of specific languages and platforms (Shadbolt 2016: 72).

Integrated Development Environments

Some Integrated Development Environments (IDEs) are designed for developing specific applications (games, web development, mobile platforms) but most support a wide range of languages and applications. IDEs are preferred for their features such as IntelliSense (Microsoft's^{TM} facility for auto completion of programming code in the Visual Studio^{TM} IDE) and the automated production of design documents and test cases, which are not present in basic text-based and command line code editors. Nevertheless, the complexity of industrial IDEs can confuse students and hinder learning. Novice programmers are reported to find error messages generated by industrial IDEs confusing, because such messages assume more advanced levels of understanding and knowledge (McIver & Conway 1996).

IDEs are regularly updated in accordance with rapidly changing market demands. However, changes to established and familiar functionality can cause confusion and frustration for students and tutors alike. Despite this, computer science departments and individual instructors tend to utilise those IDEs (and programming languages) they have greatest experience of (Pedroni & Mayer 2006). The cost of licensing of commercial software also limits choice for many CS departments.

Supporting textbooks and materials

For previous generations of university students, textbooks were the primary source of programming knowledge outside of lecture and teaching materials. In similarity with other 'static' materials, such as slide presentations, lecture notes and blackboards, any utilised textbooks communicate concepts and code snippets in a specific language (Kolling 2003; Robins et al. 2003; Norvig 2001) but often fail to adequately convey the "dynamic processes" used by experienced programmers to solve programs (East et al. 1996: 1). Soares and co-researchers (2015) offered the same introductory course to two groups; one group was required to read the course textbook, but this was not a compulsory activity for the other group. There was no noticeable difference between the assessment performances of either group.

Even though students continue to buy and read textbooks prescribed by reading lists, many utilise internet learning resources such as online courses and MOOCS, YouTube videos and recorded lecture videos, as well as blogs and forums. Educators and employers alike observe that the first instinct of student or commercial programmers is to search for a solution on Google (Utting et al. 2013: 25). Students and commercial programmers frequently read forums such as StackOverflow to inform their own solutions. StackOverflow is a question and answer site that also publishes previous discussions and code solutions.

2.3.2 Pedagogical discussion

<u>Criticisms of the conventional teaching method</u>

The teaching method of lecturing has long been utilised to communicate large amounts of information efficiently (Edwards et al. 2001; Dolnicar 2005) and is still the dominant teaching method for many subjects today. When computer science undergraduate courses were introduced during the 1960s, academics adopted the conventional teaching and assessment practices of the time (lecturing and examinations). However, critics have long argued that lecturing was an ineffective way of teaching programming (Gray et al. 1998: 94; Weinberg 1971) and this view is still held today (Freeman et al. 2014; Lister et al. 2007). Gray and colleagues (1998) comment that lecturing in a computer science education environment can lead to passive learning, premature complexity, and premature abstraction.

Studies have shown that conventional teaching methods have little impact on assessment grades (Dehnadi & Bornat 2006; McCracken et al. 2001). Dehnadi and Bornat (2006) saw that assessment grades prior to several weeks teaching at the start of the course were no different to assessment grades based on the same concepts after teaching. McCracken and co-researchers (2001) devised an experiment which required students from multiple institutions to code a simple calculator application. Even though the researchers told the lecturers not to intervene or demonstrate how to solve the task, one instructor did teach his students how to code a calculator before distributing the task (Guzdial 2016: 23). However, this group performed no better on the task than the other groups which were not taught.

Guzdial (2016: 11) outlines two alternative teaching perspectives to the conventional lecturing method. An instructor adopting the 'apprenticeship perspective' demonstrates the 'dynamic' problem-solving process (which involves the application of programming concepts) instead of transmitting concepts to them through 'static' materials (textbooks, lecture slides etc). Bennedsen and Caspersen (2005) discovered that 'exposing' the problem-solving process (verbalising their thought process) is effective for helping students who struggle to know where to start when presented with a problem.

The other teaching perspective outlined by Guzdial (2016: 11) is the 'developmental perspective'. This perspective applies constructivist learning philosophy, considering previous programming experiences and current understanding when tailoring instruction to individual student needs. However, because one-to-one tuition is "unrealistic" for most class sizes, Guzdial (2016) suggests the incorporation of opportunities for peer learning and other active learning techniques.

Pedagogies that encourage active learning

In addition to alternative teaching perspectives, specific pedagogical techniques have been developed to improve student learning in programming courses. Guzdial (2016) describes four pedagogical techniques where educators reported significant benefits from inspiring active learning amongst students:

- Peer Instruction – students are asked multiple choice questions in class and discuss the questions with their peers. Multiple studies by Porter (Porter et al. 2011; Porter et al. 2013a; Porter & Simon 2013) suggest that retention was improved and failure rates reduced as a result of the technique being used in multiple classes.
- Pair Programming – involves two students working together on a single computer to complete an exercise. McDowell and Porter's work (McDowell et al. 2002; 2006; Porter et al. 2013b) showed that this technique had positive benefits on student learning and retention.
- Worked Examples – providing students with coded examples so allowing them to identify how a problem was solved (Caspersen et al. 2008; Caspersen & Kolling 2009; Margulieux et al. 2013; Morrison et al. 2015; Sweller 1988).
- Games-Based Learning – educational games or game environments have been reported to have significant benefits for the teaching and learning of programming (Lee & Ko 2011; Mather 2014), leading to high levels of engagement and attention to detail.

Peer Instruction

The Peer Instruction method was pioneered by Eric Mazur, a physics lecturer from Harvard University, and was developed throughout the 1990s (Crouch & Mazur 2001). It was established in response to physics education research in the 1980s and 1990s, which concluded that traditional lectures did little to improve students' understanding of concepts central to the subject (Halloun & Hestenes 1985; Hake 1998; Mazur 1997: 2; McDermott 1991). Eric Mazur and colleagues wanted to create an instructional technique that would engage students actively in the process of learning physics concepts and problem-solving techniques.

The Peer Instruction method introduces multiple choice questions to the lecture. The questions are placed at different intervals in each lecture, and are often based on the key points being made (Crouch et al. 2007). Students are asked to answer individually first (initially by reporting to the lecturer, but modern implementations use 'clickers' or 'classroom response systems', texting, and social media) before discussing the question with their peers (often two or three students) and answering again. During the peer discussion, the instructor encourages students to explain their answer and vocalise the reasoning that underpins why they think their answer is the correct one (Crouch & Mazur 2001: 970). Even if students choose the same answer, they may do so for different underlying reasons (Simon & Cutts 2012a). After the students have answered again, post peer discussion, the instructor leads a class wide discussion on that question and responds in accordance with how students voted.

Students are expected to complete assigned tasks (reading and completing a quiz or listening to/watching a lecture) prior to the modified lecture to enable them to participate in group and class discussions (Crouch & Mazur 2001). Reading quizzes, which usually involve two to three questions on the reading content, and an open-ended question which asks students to describe what they found interesting or confusing (Crouch & Mazur 2001; Crouch et al. 2007). Instructors then use this feedback to further tailor the content and questions asked in class (Simon & Cutts 2012a).

<u>Educational benefits of using Peer Instruction</u>

Porter and colleagues (2013a) conducted a retrospective study of 10 years of teaching programming with Peer Instruction in 120 classes (totalling more than 10,000 students) composed of four courses (CS1, CS1.5, Theory, and Architectures). In total, 2,068 students were taught using Peer Instruction (PI), and 8612 were taught using Standard Instruction (SI). On average, the implementation of PI reduced failure rate by 61%. For CS1 (which had 1,296 PI students and 1,764 SI, specifically, there was a 59% reduction in failure rates: SI students had a failure rate of 24% whereas PI students had a failure rate of 10%. Despite raising concerns about the validity of the comparisons, the authors thought it unlikely that this reduction was a result of student improvement over time on each course, or due to PI being implemented by 'better instructors' (Porter et al. 2013a: 1).

Porter and colleagues (2016) surveyed seven different introductory programming courses taught by PI, at different institutions. Four of the seven courses were taught by instructors who were experienced with the PI technique, whereas the other three courses were taught by instructors new to the format. The results indicated high student satisfaction with the teaching format, and 71% of students would recommend other instructors to use this approach. All seven instructors (including the three unfamiliar with PI) reported that they would continue to teach their course using PI.

Other research that introduced PI to CS classes focused on the learning gains between solo and group votes of a Multiple-Choice Questionnaire (MCQ) (Simon et al. 2010; Zingaro 2010). They utilised the normalised gain metric (NG) which captures the "proportion of students who answer incorrectly in the solo vote but correctly in the group vote" (Zingaro 2010: 5031). Simon and co-researchers (2010) reported a 41% gain and Zingaro (2015) reported a 29% gain.

However, Zingaro (2015) cautions that there is potential for students to passively copy their peers' answers without attempting to understand – thereby artificially inflating the 'gain' metric for PI delivery. Although Porter and colleagues (2016) did not give students isomorphic questions (pairs of questions which test the same concept but are worded differently), they still reported learning gains from peer discussion activities.

Other educators report similar learning benefits from implementing PI in chemistry, biology, and physics courses (Crouch & Mazur 2001: 975; Freeman et al. 2007; Knight & Wood 2005).

Criticisms of Peer Instruction

Whilst developing the Peer Instruction technique throughout the 1990s, Crouch and Mazur (2001) noticed that some physics students were resistant to being taught in a non-traditional manner. This is understandable given there was little variation from didactic modes of instruction then. However, recent research suggests that students welcome non-conventional teaching techniques if these approaches will benefit their learning and the students are more willing to engage with them as a result (Porter et al. 2016: 361).

However, Porter and colleagues' (2016) multi-institutional study revealed one instance with an abnormally high portion of dissatisfied students. It later emerged that the dissatisfied students were from the same class which was taught by an inexperienced instructor. This instructor did not fully explain the use of 'clickers' (according to 25% of the class); an issue that had previously been found to lead to confusion and disengagement with the PI technique (Duncan 2008). The instructor also graded their in-class questions, which is not advised by Porter and colleagues (2016: 362) because the questions were intended to be formative and a "tool to build [student] understanding" instead of being viewed as a 'test'. Student learning is hindered when an incorrect answer provided during the formative learning process has implications for their summative grade.

The implications of unclear expectations and structural issues were revealed in the survey responses returned by students (Porter et al. 2016), as students felt that:

- the clicker questions were too hard;
- they did not have enough time to answer the questions;
- they did not always choose to discuss with their peers;
- they did not find value in hearing other students explain their answers or the class wide discussions; and
- they would ascribe less value to PI for their learning.

Despite specifying guidelines for the implementation of PI (Porter et al. 2013), it is difficult to ensure consistency due to unique class dynamics or teaching styles. Some teachers do not incorporate reading quizzes (Zingaro 2014b) and there is little consistency in curriculum design.

Although PI improves satisfaction ratings and retention for students, the impact on exam results is contested. Zingaro (2014a: ii) found a 4.4% difference between PI and SI students, which is substantial in terms of grades, but a non-significant statistical result means that "similar gains cannot be expected in other situations". However, Simon and co-researchers (2013) found that PI students scored on average 5.7% higher than students receiving conventional lectures.

Pair Programming

In contrast to the pedagogies developed specifically for teaching programming, Pair Programming is a technique that originates from the software development industry and has now been adopted by educators. Software development companies have long supported teamwork amongst programmers. However, the Pair Programming technique prescribes a unique approach to collaboration. This requires two programmers to share one keyboard and one terminal. The programmers alternate between defined roles. One person is in control of programming (the 'driver') and the other will 'navigate' the driver (the latter participant using their time to think ahead and plan). One of the pioneers of the approach, Kent Beck (2000), suggests that programmers swap roles every 45 minutes but employees and instructors often set and adjust working formats depending on the project.

There is significant evidence that this method increased productivity amongst employees in industry. Nosek (1998) gave 15 experienced full-time programmers a challenging problem (but relevant to their organisation) to attempt in their familiar working environment. Ten programmers were paired up to work on the task, and the other five were asked to work alone. The pairs spent more time on task (being 60% more productive), completed the tasks 40% more quickly and produced better quality algorithms and code than the individual programmers.

Although most of the participants were initially hesitant about working together, afterwards, they not only enjoyed the problem-solving process more, but they also had greater confidence in their solutions (Nosek 1998). Similarly, Constantine (1995) observed that pairs of commercial programmers produced code more quickly and with fewer errors. Recent employee surveys maintain the positive benefits of appropriate pairing - increased productivity and creativity, better understanding of the project they are contributing towards, and developing closer relationships with their employees through working together (Begel & Nagappan 2008; Vanhanen & Lassenius 2007).

Educational benefits of using Pair Programming

Laurie Williams was one of the first to trial Pair Programming in a university class. She noticed that paired students produced better quality solutions and also learned subject content quicker than those working alone (Williams et al. 2000; Williams & Kessler 1999). Similarly, McDowell and co-researchers (2006) noticed an increased motivation to program and a commitment to continue studying computer science as a result. Braught and colleagues (2008) found that students with lower entrance exam grades experienced the greatest learning gains as a result of participating with Pair Programming. Similar studies found that engagement with collaborative learning tasks reduced likelihood of withdrawal from and/or failing the course (Gehringer et al. 2006; Maguire et al. 2014).

Williams (2007) conducted seven years of Pair Programming research and consistently observed strong relationships being built between pairs of people through working together. Stronger relationships lead to the development of belonging and comfort, which in turn may stimulate learning (Section 2.2.4). Students also report feeling more confident from programming in pairs. Many have reported Pair Programming to be highly engaging for students, allowing them to spend more time on task and often completing the task more quickly than when working individually. Pairs are less susceptible to procrastination than those working individually (Avila 2016), as they motivate each other to stay on task – sometimes referred to as 'pair pressure' (McKinney & Denton 2006).

Students and employees also reported increased creativity through working together when surveyed on their experience of using Pair Programming. Even pairing of experienced programmers has often resulted in unexpected innovation through combining expertise (Williams 2007). Encouraging students to collaborate not only helps them to learn together but prepares them for the type of collaborative working environments found in industry (Williams 2006).

Implementations of Pair Programming in education tend to involve shorter time intervals for exchanging roles than Beck's original 45-minute recommendation. The intervals range from 12 minutes (Braught et al. 2010) to 20 minutes (Wood et al. 2013) in educational settings so that students may experience both roles. In addition to MacGregor's (1988) observations, Williams (2007) noticed that Pair Programming allows students to resolve questions amongst themselves without intervention from the instructor.

There is scope for capturing student interaction in the Pair Programming process on video. Donna Teague (2011: 117) recommends recording the verbal and physical interactions of students in order to gain valuable insights around how those students resolve issues, and how they develop correct and incorrect understanding of programming concepts.

Criticisms of Pair Programming

The process of selecting pairs is commonly discussed in Pair Programming research. In industry, it was noticed that pairing expert programmers with inexperienced programmers was only beneficial if the expert took on a mentoring role. Difficulties occur when experts are unwilling to relinquish their 'driving' position, leaving the inexperienced programmer to observe them complete the work (Stephens & Rosenberg 2003). However, if an expert is willing to invest time in training the inexperienced programmer, it is more likely to increase the long-term productivity of the company.

In educational settings, researchers have encountered similar issues when pairing students with different levels of programming knowledge and experience. Teague (2011: 117) conducted a longitudinal study of 34 empirical articles published between 1998 and 2010, finding that "appropriate pair-selection was crucial to success". Stephens and Rosenberg (2003) discovered that pairing programmers of similar nature and ability was the most beneficial arrangement (assuming they had a certain level of programming ability). However, Preston (2006) noticed that inexperienced students can benefit from an experienced student willing to mentor and explain ideas to them. This has similarly been observed in studies conducted in industry (Stephens & Rosenberg 2003). The experienced student also reinforces his or her knowledge and understanding by having to teach concepts and strategies. In contrast to the observations of Stephens and Rosenberg (2003), Sfetsos and co-researchers (2009) discovered that pairs with dissimilar temperaments produced work to a higher standard. Contradictory findings, however, may indicate that outcomes depend more on the working arrangement of the two students than their personalities or experience. As discussed earlier, the learning and working approaches of students are significantly influenced by their motivations and goals for studying (Section 2.1.3 & Section 2.2.2).

Pair Programming was found to be ineffective in environments where students did not engage with the process or did not attend consistently (Bevan et al. 2002; Han & Basheti 2010; Williams et al. 2008). In response, researchers recommended implementing a strict attendance policy in order to protect students from non-participating partners (Bevan et al. 2002; Williams et al. 2008). However, as previously stated in Section 2.2.1, opponents of mandatory attendance policies argue that students should not be forced to attend if they do not wish to (Grove 2016; Feldman 2013). Instead of enforcing a mandatory attendance policy, other researchers recommend modifying assessments to ensure that grades do not solely depend on collaborative work, thereby protecting students from being disadvantaged from disengaged peers (Williams et al. 2008). Other educators have reported difficulty in creating meaningful exercises which are both sufficiently complex to require two programmers, but also solvable within the time available (Kolling & Barnes 2008).

Similar to PI, unclear explanations of requirements and reasons for using a collaborative method lead to confusion and disengagement with the working format (Bevan et al. 2002; Han & Basheti 2010; Williams et al. 2008). Researchers recommend active monitoring of student progress in order to respond to issues early (Han & Basheti 2010; Simon & Cutts 2012b).

Radermacher and colleagues (2012) set out to investigate whether Pair Programming could improve mental model consistency (MMC) amongst students holding inconsistent models but they were unable to find a pair arrangement that improved consistency. Porter and colleagues (2013) observed little difference between the exam results of paired and unpaired students. However, more paired students completed the course and took the final exam than those working alone. Therefore, whilst many report that Pair Programming improves the quality of, and enables quicker completion of work, it may not necessarily lead to an improved understanding of programming concepts.

Worked Examples

In contrast to the Pair Programming and those Peer Instruction techniques that encourage students to collaborate in order to solve problems, the provision of Worked Examples aims to reduce the cognitive strain of problem solving for individuals. Conventional instruction in computer programming assumes that students improve their problem-solving ability by working through many problems (also known as the problem-based approach). Novices are therefore often given problems to solve with minimal guidance, which may subsequently result in confusion and frustration. Kirschner and colleagues (2006) argue that guided models of instruction are likely to benefit the learner more than unguided models of instruction. Worked Examples are a form of guided instruction that provide a complete or partially complete solution for students to read and study.

The process of problem-solving is recognised by all STEM disciplines to require significant amounts of cognition. John Sweller (1988) introduced the Cognitive Load Theory (CLT), in which he describes ‘cognitive load’ to represent the amount of information held in the brain’s working memory that is required for the problem-solving process. However, if the information required to be held exceeds the capacity of the working memory, this may lead to an ‘overload’. Learning is hindered if students cannot handle all the information they require to solve a problem.

Cognitive load is regarded as falling into three separate capacities: intrinsic cognitive load (the difficulty of the content being taught); extraneous cognitive load (how the material is taught); and germane cognitive load (committal of knowledge to long term memory and the formation of mental models). It is possible to reduce intrinsic and extraneous cognitive load and to increase germane cognitive load through instructional design (Margulieux et al. 2012; Sweller 2010).

Definitions of 'Worked Examples' vary in the literature, but Caspersen and colleagues (2008) have noticed that most definitions share two common attributes. Worked Examples tend to include a problem statement and a procedure for solving the problem (Caspersen et al. 2008: 870). The examples provide a template for the solution so that students become familiar with the patterns associated with solving related problems (Skudder & Luxton-Reily 2014). This instructional technique can be considered a response to researchers' requests to explicitly teach and demonstrate strategies for composing solutions to programming problems (Bennedsen & Caspersen 2007; Caspersen & Kolling 2009; de Raadt et al. 2004). Worked Examples provide the full (or in some cases partial) solution in a step by step format (Clark et al. 2011; Caspersen et al. 2008), with the purpose of reducing the cognitive strain often attributed to the problem-solving process. Some include diagrams illustrating design elements of the solution (van Gog et al. 2004) whilst others present the final answer (Atkinson et al. 2003).

Educational benefits of using Worked Examples

A number of researchers in computer science and programming courses report reductions in cognitive load through using Worked Examples (Caspersen & Kolling 2009; Gray et al. 2007; Gregory et al. 1993). Worked Examples have been discovered to be effective in reducing cognitive load during the initial weeks (where students are most vulnerable to cognitive overload) in a variety of courses (Clark et al. 2006; Sweller et al. 2011). Recent work has extended Worked Examples by providing 'subgoal labels' (Margulieux et al. 2012; Morrison et al. 2015; Schaeffer et al. 2016). Subgoal labels provide additional information alongside other core requirements for programming tasks, similar to the information an instructor would provide to students who request further guidance. The purpose of labelling subgoals is to help learners recognise the structure of examples.

Margulieux and colleagues (2012) compared the relative performance of programming students who were provided Worked Examples with subgoal labels, and students who were provided with Worked Examples without subgoal labels. The authors found that the provision of subgoal labels reduced cognitive load and helped students to create accurate mental models of concepts. The creation of mental models further reduces the cognitive load required to process new information (Kirschner et al. 2006).

As stated in Section 2.2, expert programmers typically approach a problem in a structured manner, utilising and reworking previous solutions. Since novice programmers have few or no previous solutions to draw from, having access to the full solution provides a foundation to overcome confusion of knowing where to begin (Clancy & Linn 1990).

Despite their lack of experience, Fincher (1999) found that most novice programmers are capable of reading much more complex programs than they can write. By studying code, students become familiar with how certain tasks and actions are coded. Being provided with well-structured and well-written code will also instil good design and practice that is desired by both educators and industry. Such examples also help inspire confidence that solutions are achievable and often less complicated than expected.

Skudder and Luxton-Reily (2014) define four specific types of Worked Examples utilised in other related science and mathematics subjects:

- examples only: Worked Examples without associated exercises or problems to solve;
- example-problem blocks: students are provided with a set of Worked Examples, then a set of exercises to solve;
- example-problem pairs: each worked example is paired with problem to solve (that is similar to the worked example); and
- faded Worked Examples: students are initially presented with complete Worked Examples, but as the course progresses, more steps are removed from these examples each time until students are just presented with a problem.

Studies show that students who received Worked Examples alongside a similar practice problem to solve (example-problem pairs) outperformed students who received blocks of Worked Examples separately from blocks of practice problems (Trafton & Reiser 1993; van Gog et al. 2011). However, van Gog and colleagues (2011) discovered little difference between the performance of students given 'example pairs' and those only provided with an example ('example only'). Fading the Worked Examples, by progressively removing solution steps from examples, helps students to make the transition to problem-solving. There are regarded to be two separate types of fading: forward fading (which removes the steps from the top of the code first) and backward fading (removes steps from the end of the code first). Moreno and colleagues (2006) found 'forward fading' to be more effective than 'backward fading'.

A recent study found self-explanation prompts (questions asking students to explain the steps they took in order to solve a problem) to be effective in reinforcing understanding of concepts when combined with Worked Examples (Atkinson et al. 2003; Booth et al. 2015). Skudder and Luxton-Reily (2014: 60) write that self-explanation prompts have the potential "to be a source of germane cognitive load" which thereby enforces committal to long term memory.

The combination of studying a worked example, answering self-explanation prompts, and then solving a related problem, is an effective way to reinforce understanding of algebra concepts (Atkinson et al. 2000; Atkinson et al. 2003; Davidson 2011: 30; Trafton & Reiser 1993). This approach also enables students to solve algebra problems with fewer errors and less teacher intervention as a result (Carroll 1994; Sweller & Cooper 1985). On the other hand, providing incorrect Worked Examples has similarly helped students recognise incorrect assumptions and procedures in the problem-solving process. Students gain both conceptual and procedural knowledge from comparing incorrect and correct Worked Examples (Booth et al. 2013).

Criticisms of the Worked Examples

One disadvantage of using Worked Examples is that students can become dependent on receiving complete solutions and then do not develop their own problem-solving abilities. Providing too much support can be as detrimental as providing too little (Margulieux & Catrambone 2016). The gradual reduction in support that is characteristic of 'faded Worked Examples' is an effective method of scaffolding the development of problem-solving techniques. However, more experienced students did not benefit from examples intended for novice students, and instead required examples appropriate for their experience (Kalyuga et al. 2003; Skudder & Luxton-Reily 2014: 63).

Eiriksdottir and Catrambone (2011) argue that Worked Examples do not necessarily promote deep learning of concepts. Novices read and study the completed solutions but do not have experience and domain knowledge to know which solution features are most significant (Teague et al 2013). Whilst the examples may aid exercise completion and recognition of patterns, students may not use them to develop their understanding of programming principles. As discussed earlier, goals are closely linked to motivation (Section 2.1.3 & Section 2.2.2). Therefore, the impact of tools and interventions may be limited if students are concerned with different outcomes. However, Biggs and Tang (2011) discovered that student learning approaches did change when aspects of the course were constructively aligned and promoted active learning. This is further discussed in Section 2.3.

The application of Worked Examples in subject domains such as physics and mathematics is well documented. However, there is an opportunity to evaluate the effectiveness of Worked Examples in computer science contexts, beyond the specific extension of 'subgoal labels'. Despite the positive benefits being reported by Margulieux and colleagues (2012), Morrison and co-workers (2015) discovered little difference in assessment performance between those people provided with subgoal lables and those expected to generate them. The authors do not discount the importance of mental model creations but suggest that additional support is required for these to be effective.

Games-Based Learning and Gamification

Educators have attempted to improve engagement with learning activities by involving elements of gameplay (Livingstone 2011; Davidson 2011). Recent studies indicate that video games are maintaining their widespread appeal as they are played by nearly 99% of 8-15 year olds and 60% of adults (Internet Advertising Beureu 2014; Kahne 2008; NESTA 2015: 18).

The literature refers to the use of games in learning activities by different definitions:

- Game-Based Learning: using video games to encourage learning (Perrotta et al. 2013);
- Gamification: incorporating game aspects and elements into conventional teaching practice (Faiella & Ricciardi 2015);
- Gameplay: aspects of the course become game elements – topics and ideas become "rules, actions, decisions and consequences, rather than content to be communicated" (Livingstone & Saeed 2017: 91).

Despite the different categorisations and levels of incorporation with learning environments, the skills required for computer programming overlap with the skills required for game play (Livingstone & Saaed 2017). Games and programming require problem-solving and logical skills. Games are also created by programmers (and creative artists) and can therefore inspire those wishing to work in the games industry.

Since the early 2000s, gaming environments and interactive platforms have been developed specifically to teach programming skills and concepts. Some notable languages and platforms include 'Alice' (Cooper et al. 2000), 'Jeliot' (Ben-Bassat Levy et al. 2003) and 'Scratch' (Maloney et al. 2003). These were created specifically for visualising programming execution. The recent drives to engage young people with coding in the UK and US (as explained in Section 1.2) has led to the creation of web-based educational games such as Gidget, TryObjectiveC, code.org and codecombat.com. Code.org has partnered with popular games such as Minecraft and Flappy Bird, as well as popular feature films such as Star Wars and Frozen to deliver relevant and engaging learning contexts.

The specific tasks associated with each game-oriented learning environment vary but most educational games require students to program correct solutions to complete tasks. In contrast to non-educational games, educational games attempt to make the problem-solving process explicit and teach programming concepts through developing solutions.

The educational benefits of using games to teach programming

Perrotta and colleagues' (2013: 10) review found that the use of educational video games improved problem-solving skills, motivation and engagement in most instances. Educational games are often designed to be collaborative (encouraging workplace skills), and most feedback has indicated that students enjoyed learning through games (Livingstone & Saeed 2017: 92; Fotaris et al. 2015).

Fotaris and colleagues (2015) replaced their conventional hour-length lectures with three cycles of a shorter 20-minute lecture followed by a multiple-choice formative assessment game ('Kahoot') and a short discussion of the quiz results. Seminars were also modified to include an adaptation of the 'Who Wants to be a Millionaire' multiple choice game, which asked questions on programming concepts and the Python programming language. In a later survey, over 75% of students strongly agreed that the game activities made the learning environment a fun and engaging context to learn. Over 50% agreed the approach improved their analytical and problem-solving skills and student attendance improved by 40%.

Sung and colleagues (2010: 8) found that students who received game-themed assessments (GTAs) achieved, on average, a grade higher than those receiving conventional assessments. They also noticed that the difference between average grades was greater for later assignments in the introductory programming course (Sung et al. 2010: 8).

Emerging neuropsychological research also provides scientific evidence for the learning benefits of using games for educational purposes. Kuhn and co-researchers (2014) set out to monitor changes in the neural structure of the brain during and after prolonged interaction with video games using magnetic resonance imaging (MRI). The researchers asked 23 of the 48 German participants (all of whom were not avid gamers) to play the video game 'Super Mario 64' (on a Nintendo DS gaming console) for at least 30 minutes a day, over two months. The other 25 participants acted as the control group (they did not play the game) but still participated in the MRI brain scans with the experimental group. The scans revealed "significant gray matter (GM) increase in right hippocampal formation (HC), right dorsolateral prefrontal cortex (DLPFC) and bilateral cerebellum" regions of the brains of the experimental group (Kuhn et al. 2014: 265). These regions of the brain are responsible for spatial navigation, working memory, strategic planning and fine-motor skills. The increased neural activity in these regions strengthens the pathways between them, often associated with learning or repeated behaviours (Davidson 2011: 46). Kuhn and colleagues (2014) also discovered a positive correlation between participants that most wanted to play the game and the scans indicating the greatest 'gray matter' increase.

Educational games can also provide a concrete and visible expression of abstract and 'invisible' programming concepts. Students can associate each line of code with perceptible behaviour in the game environment. This has the potential to inform mental model constructions of programming concepts. Associating gameplay with learning within the classroom is also likely to encourage learning when students play games outside of the classroom (Davidson 2011; Livingstone & Saeed 2017: 92).

Criticisms of game-based learning

However, a review carried out by Perrotta and colleagues' (2013: 10) discovered contrasting results regarding the impact on academic achievement. Following test questions based on sequence, selection and iteration concepts, Barnes and colleagues (2007: 4) found little difference between the pre-test and post-test scores of students who played 40 minutes of educational games. Yet despite the lack of improvement in test scores, students were still able to complete the game objectives. Task completion does not necessarily align with understanding of programming concepts. Mather (2014) observed that students could achieve high grades on their assessed work (completing tasks in a robotic game environment) but still fail to answer basic questions about the underlying programming concepts.

It is also possible that game objectives may distract students from thinking about programming concepts and developing problem-solving techniques. Perrotta and co-researchers (2013) recommend embedding game activities within a defined pedagogic process and ensuring that students are told the expectations for using games in a learning environment. As previously found with PI and PP research, clearly communicating the reasons for using a non-conventional approach helps to reduce confusion and disengagement (Bevan et al. 2002; Han & Basheti 2010; Porter et al. 2013a; Williams et al. 2008).

The same criticisms made of 'safer' and 'novice-friendly' programming languages can also be applied to Games-Based Learning approaches. Some workers believe that these learning formats do not adequately prepare students for industrial working environments given that few workplace settings (except for game development organisations) allow employees to conduct their work through gaming activities. Therefore, educators are hesitant to recommend teaching approaches that are solely dependent on gaming activities (Perrotta et al. 2013: 10). Nevertheless, most educators indicated that they would be willing to incorporate aspects of gameplay alongside their existing teaching practice as a means to increase engagement (Perrotta et al. 2013: 10).

Comparing traditional instruction with pedagogies that encourage active learning

The above review of four popular examples of pedagogies that encourage active learning (Peer Instruction, Pair Programming, Worked Examples and Games-Based Learning) has identified that these all have a mixture of both merits and disadvantages. These are summarised below.

The potential benefits of innovative learning environments:

(1) Greater student engagement, satisfaction and motivation

Porter and colleagues (2016) found high student satisfaction with the Peer Instruction approach, and all tutors surveyed indicated that they would continue using the approach. There is significant evidence that Pair Programming increased productivity among commercial programmers (Constantine 1995; Nosek 1998) and in educational contexts (McDowell et al. 2006; Williams et al. 2000; Williams & Kessler 1999). Similarly, three quarters of students in the work of Fotaris and colleagues (2015) highly enjoyed the use of games in their learning activities, leading to substantial improvements in course attendance.

(2) Reduced failure rates

The Peer Instruction approach was found to reduce failure rates by 61% (Porter et al. 2013) across multiple CS courses. PI reduced failure rates in a first semester programming course by 59%. After implementation, only 10% of PI students failed the course, compared to 24% of SI (standard instruction) students. Pair Programming also reduced the withdrawal and failure rate for introductory programming courses (Gehringer et al. 2006 Maguire et al. 2014).

Freeman and colleagues (2014) conducted a meta-analysis on 225 studies reporting examination scores or failure rates for STEM subjects for both traditional lecturing (n = 67 studies) and active learning (n = 158 studies). They found that students were “1.5 times more likely to fail” if they had conventional lecturing, compared with teaching approaches that cultivated active learning.

(3) Peer teaching and learning

Different authors (Simon et al. 2010; Zingaro et al. 2010) have reported a 30-40% improvement in test scores for the Peer Instruction method, as indicated by the greater number of group vote correct answers in comparison to those for solo/individual vote tests. As group discussions only last three to five minutes, it would appear that this is a time efficient means for peers to correct any misconceptions. Likewise, pairs of students working collaboratively can learn strategies and concepts from each other rather than solely from the instructor (Davidson 2011: 109, 129; Gokhale 1995; McGregor 1988).

There is likely to be a greater difference in knowledge and experience between instructor and student, than between students. Instructors may experience the so-called “expert blind spot” (Wiggins & McTighe 2006) in that they do not recognise that some aspects of a topic that are so clearly obvious to them are less apparent to their students. Therefore, Peer Instruction and communication can improve learning gains as peers will be at similar cognitive stages (Simon et al. 2010; Wiggins & McTighe 2006; Zingaro et al. 2010).

Possible disadvantages of innovative learning environments:

(1) Mixed reports regarding real impact on academic achievement

Whilst most studies analysing the impact of active learning pedagogies report significant reduction in failure rates and therefore, an improvement in retention, the impact on academic achievement is less clear.

Porter and colleagues (2013a) noticed little difference between the exam results of paired students and those working individually. Others observed marginal improvements in the exam grades for pairs (4-5%), although the significance of some of these reported improvements has been contested (Simon et al. 2010; Zingaro 2014a). Some authors also observed little difference in assessment grades following the use of Worked Examples (Morrison et al. 2015) and Games-Based Learning approaches (Barnes et al. 2007; Perrotta et al. 2013).

(2) Mixed reviews with respect to gains in understanding

Further findings suggest that assessment grades do not necessarily reflect student levels of understanding of programming concepts. Zingaro (2015) cautions that the learning gains reported by Peer Instruction methods may result from peers sharing answers rather than from any real discussion whereby students fully explore programming concepts (Section 2.1). Although Pair Programming may lead to quicker development of solutions, there is some evidence that the technique did not improve mental model consistency amongst participants (Radermacher et al. 2012).

Similarly, students may use Worked Examples to copy correct solutions (Eiriksdottir & Catrambone 2011) or, with respect to the use of gameplay, they may become distracted such that the intended learning development does not take place (Mather 2014; Perrotta et al. 2013). This would corroborate separate findings indicating that the learning approaches and motivation of students have greater influence than the teaching methods utilised, and would also align with constructivist beliefs (Matheison 2015).

(3) The need to explain instructions clearly

A number of studies record that students who disengaged with the active learning techniques commented that instructions and/or the reasons for using the technique were not explained clearly enough (Bevan et al. 2002; Han & Basheti 2010; Porter et al. 2013a; Williams et al. 2008).

(4) Lack of time to implement large curriculum modifications

Most tutors regarded non-conventional techniques as beneficial, and would recommend them to their colleagues (Porter et al. 2013a). However, these pedagogies are not widely adopted outside of the institutions in which they were evaluated (Barker et al. 2015). Recent publications have raised awareness of a need for teaching innovation amongst educators. The introduction of a new UK school computer curriculum in 2014 (Section 1.2) has also provided an opportunity to educate teachers in new approaches (Sentance & Csizmadia 2016). Despite this, Guzdial and colleagues (2014) found that few undergraduate level educators would consider changing their teaching approach. Although acknowledging the validity of research, some educators require further evidence before adopting new practices (Perrotta et al. 2013). Others favoured traditional lecture methods because they are most familiar and accessible (Ni 2009; Wankat et al. 2002). A recent survey (UCU 2016: 6) found that, on average, academics now give twice as much time to teaching and assessment activities than they give to research activities. However, nearly half of academics indicated that they do not have time to prepare their lecture materials (Times Higher Education 2017).

2.3.3 The assessment of programming courses

Whilst assessment varies from institution to institution, introductory programming courses tend to adopt three main assessment formats. Bennedsen and Caspersen (2007: 35) found, in their survey of 63 CS1 courses, that "on average, 35% of the final grade is due to marked assignments during the course, 35% is from the final exam, and 30% is from some other source (e.g. lab-exercises, midterm exams, or programming projects)." The advantages and disadvantages of these different types of assessment are discussed below.

Laboratory exercises

Laboratory exercises are generally based on smaller and, by their nature, less complex problems. They are often supplied in practical laboratory sessions and provide active learning opportunities for students as well as feedback for tutors. Some institutions assess these exercises and others utilise them as formative activities which prepare students for larger assignments or projects (Kolling & Barnes 2004: 287).

However, small-scale laboratory programming exercises are more vulnerable to plagiarism because students can easily copy solutions from the internet or from peers (Ahadi & Matheison 2019a). The limited scope of the exercises usually means that there will be fewer alternative solutions and this tempts students to think copied attempts will be harder to identify (Luck & Joy 1998). Williams (2006) found that fewer plagiarism cases were reported in classes when Pair Programming was implemented as the method rewards students working together rather than penalising them.

Projects and assignments

Projects and assignments tend to be modelled on larger and more complex scenarios than individual laboratory exercises. They often require students to document the design, testing, and evaluation of their programs, mirroring the processes required in formal software development lifecycles (Sommerville 2010: 30). In addition to assignment and laboratory exercise submissions, some instructors ask students to 'walkthrough' their code (like an informal interview) to ensure code is understood (McCracken et al. 2001: 3).

Even though later laboratory exercises usually build upon concepts introduced in earlier weeks, in any given session exercises tend to focus upon the topics for the current week. However, assignments often require students to integrate subject knowledge over longer periods (Kolling & Barnes 2004: 287). Studies suggest that students struggle to do this (Luxton-Reily et al. 2017; McCracken et al. 2001), even if they are able to successfully answer questions on isolated concepts or individual topics (Luxton-Reily et al. 2017).

Exams

Computer science courses adopted the examination assessment technique common to other disciplines during the 1960s at a time when undergraduate programmes were being formalised (Lloyd 2013; Jones 1999; Navarro-Lopez 2014). Most introductory programming courses measure learning progress by examinations at the end of the curriculum and some also feature intermediary mid-term exams too. Literature indicates that multiple choice questions (MC) and constructed response questions (CR) are preferred formats for programming examinations. Although online MC exams may offer the convenience of automatic marking, MC questions tend to assess different skills to CR questions (Davidson 2011: 114). MC questions suit the assessment of factual content with defined correct answers (Davidson 2011: 111), whereas, CR questions require students to explain their understanding of concepts and programming scenarios in their own words. The CR format is therefore regarded to more accurately assess 'higher-order' thinking skills (similar to essays) and understanding (Davidson 2011: 114 - 115).

Both MC and CR responses are still vulnerable to memorisation techniques (associated with the surface learning approach) and plagiarism (Luck & Joy 1998). However, Azalov and colleagues (2004) discovered that CR assessment was more strongly correlated with final grades for an undergraduate introductory CS course than a related MC assessment.

Parsons and colleagues (2015) highlighted past studies (Lister et al. 2004; Dehnadi & Bornat 2009) that gave students deliberately complicated and unrealistic code fragments to trace for exam questions. These questions are not logical and not representative of the types of scenarios that commercial programmers work to solve in industry. Incorrect answers to such questions cause educators to wrongly assume that students cannot code (Parsons et al. 2015).

Formative Assessment

The above three types of assessment are typically used as summative instruments towards final grades (Bennedsen & Caspersen 2007) rather than for formative assessment which is primarily to support learning (Lilley & Barker 2007). When there is less emphasis on grades and marks, this removes much of the distraction and pressure for students and also reduces cognitive load (Margulieux et al. 2012; Sweller et al. 2011). This in turn enables students to concentrate on their development (Gibbs 2015). Gibbs (2015) goes further to suggest that limiting summative assessment to simple pass/fail options would remove much of the unnecessary pressure that students face during their first semester.

Formative assessment types widely range from quizzes (University of Exeter n.d.; Radenski 2007; Roberts 2006; Denny et al. 2008) to game activities (Fotaris et al. 2015). Most applications of formative activities report benefits in learning gain and increased student engagement. Feedback from formative activities also informs modifications to teaching content and delivery (OECD 2005: 25). Corbett and Anderson (2001) discovered feedback to be most effective when it is communicated immediately after the unassessed task and addresses specific steps and features in student work. However, feedback is only effective to the extent that students respond and adjust accordingly (Gibbs 2015; Koulouri et al. 2014: 25).

Summary of Section 2.3

This section considers literature concerned with the design and delivery of introductory programming courses.

Studies concerning the components that form programming course environments (Section 2.3.1) suggest that students in small class sizes performed better than those in larger classes (Bennedsen & Caspersen 2007). This is thought to be due to increased opportunities for the collaborative and peer learning behaviours that are discussed in Section 2.3.2. Section 2.3.1 also demonstrated a tension between programming languages and the environments designed to be accessible to novices, and the sophistication of tools used in the workplace which, because of their complexity, may hinder introductory-level learning (Jenkins 2001a; McIver & Conway 1996). Industry requires students to possess fundamental and transferable skills in order that they may adapt to changes in the workplace (Section 1.4). Supplementary textbooks were reported not to be particularly helpful for imparting the problem-solving techniques, which, in Sections 2.1.1 and 2.2.2 were found to be a strong predictor of programming success. Nevertheless, in the place of textbooks students increasingly consult alternative online sources for programming help (Utting et al. 2013).

Section 2.3.2 discussed recent developments in pedagogical approaches to teaching programming which are designed to encourage active learning. Such techniques are associated with reducing failure rates (Freeman et al. 2014 Porter et al. 2013a), increasing engagement and motivation (Constantine 1995; Fotaris et al. 2015; McDowell et al. 2006; Nosek 1998; Williams & Kessler 1999), as well as creating efficient and effective peer learning opportunities (Simon et al. 2010; Zingaro et al. 2010), due to students developing similar levels of cognition (Wiggins & McTighe 2006). However, whilst students may be able to develop solutions in less time, this process may not lead to greater and deeper understanding of programming concepts (Eiriksdottir & Catrambone 2011; Mather 2014; Radermacher et al. 2012; Zingaro 2014b). The extent to which learning goals and outcomes are met depends greatly on motivation, which in turn informs student learning approaches and priorities (Section 2.2.2 & Section 2.1.3). Despite significant benefits reported for innovative teaching practices, educators required more evidence before being willing to adopt new techniques (Perrotta et al. 2013). Educators who were comfortable with familiar teaching methods and materials were often unwilling to make major changes to their courses (Ni 2009; Reis 1998). However, some educators were more willing to make minor adjustments. Changes to teaching approaches must also be clearly communicated to students if confusion and disengagement are to be avoided (Bevan et al. 2002; Han & Basheti 2010; Porter et al. 2013a; Williams et al. 2008).

Section 2.3.3 reviewed techniques for assessing programming ability. Laboratory exercises introduce problem-solving in a small and manageable context but are more susceptible to plagiarism. Although assignments and projects provide opportunities to set more complex and challenging problems, most novices have difficulty applying multiple programming concepts and understanding the relationships between concepts (Luxton-Reily et al. 2017). Regarding examinations, constructed response (CR) exam questions proved to be better at predicting final grades than multiple choice (MC) questions (Azalov et al. 2004). However, exam questions can be unrepresentative of industry problems (Barker et al. 2015), and may also misrepresent real coding ability in the workplace. The fact that students also express a fear of exams, known as test anxiety, which can detrimentally affect learning performance, was considered in Section 2.4.

2.4 Discussion and identification of research opportunities

This review explored three areas which were thought to influence learning outcomes and student experiences associated with programming courses.

Section 2.1 was concerned with research exploring the effects of entry qualifications, student profile characteristics and motivations for study. A key finding was that A-level Computing qualifications generally better prepared students for undergraduate computer programming courses than A-level IT courses due to greater coverage of programming concepts (Boyle et al. 2002; Drummond 2009). However, studies reviewed in Section 2.2 (addressing attendance and engagement with course, learning approaches and learning styles, technology habits and emotional welfare) indicated that problem-solving ability explicitly taught in discrete mathematics courses (Pioro 2006) was also found to beneficially influence programming ability (Section 2.2.2). Workers also expressed doubts that conventional teaching methods and resources were effective for communicating problem solving skills. The third area for attention, Section 2.3, teaching methods, pedagogies and assessment techniques highlighted that approaches designed to communicate required skills by utilising peer interaction and collaborative behaviours, can lead to greater engagement, retention, and efficient task completion and learning opportunities (Section 2.3.2). Moreover, motivations tend to inform learning approaches and these approaches determine how a student engages with teaching material and techniques (Section 2.1.3). Constructivist philosophy believes that students determine what is learnt (Matheison 2015) but appropriate and aligned course elements can influence their use of deep and surface learning approaches (Biggs & Tang 2011). An emphasis on summative assessment will reinforce the importance of grades, but well-designed formative assessment and feedback can direct attention towards the development of one’s skills and abilities (Gibbs 2015).

This review has highlighted a need to further understand and investigate the following three factors as potential influences of learning outcomes and experiences:

1. BTEC qualifications: earlier studies have not evaluated the effectiveness of BTEC qualifications because their widespread uptake for university entrance is a relatively recent phenomenon. The recent rise in BTEC applicants and the very different educational circumstances that surround BTEC and A-level has created a need to more fully understand learning performance and other consequences associated with these two paths for university entrance.
2. Pair Programming and peer interaction: collaborative techniques have been comprehensively researched using conventional CSEd data collection methods, but there is relatively little understood concerning the detailed behaviours and tacit communications that take place between participants. There is a strong case to use video analysis to explore Pair Programming and peer interaction and Teague (2011) recommends using this technique to discover the detailed interactions that may be overlooked or missed by other forms of investigation.
3. Teaching design: there is a need to identify examples of best teaching practice (Gordon 2016) which align course elements to encourage deep learning of programming concepts. The use of formative assessment should be explored for the ability to create opportunities for personal development (Gibbs 2015). Teaching methods that communicate problem-solving processes and create efficient learning opportunities, which involve peer participation, will also be investigated. However, teaching interventions that make incremental changes that build on existing practice are more likely to be adopted by other educators (Barker et al. 2015; Perrotta et al. 2013).

This chapter addresses Objective 1 “*Review predictive trends and student behaviours already identified in the literature*” and Objective 2 “*Investigate pedagogical interventions that have been implemented in other higher education establishments*”, as stated in Section 1.5. Chapter 2 also establishes research priorities so that appropriate methodology and design of research studies may be considered in Chapter 3.

Chapter 3 Methodology

This chapter outlines the research approach adopted for the studies reported in this thesis. The methodology was informed by discussion with the research community and by reviewing computer science education literature. Computer science education (CSEd) research is introduced in Section 3.1. This section summarises categories of CSEd research and also considers the challenges of maintaining research quality and confidence in the research findings for this field. Section 3.2 outlines the methods commonly reported in computer science education research publications, including research approaches, methods of data collection, sample sizes and commonly used techniques of data analysis. Section 3.3 draws on the earlier two sections to provide a rationale and explanation for the research approach adopted in this thesis and the design of studies conducted.

3.1 Introduction to Computer Science Education research

Research in computer science (CS) education is relatively young compared with other scientific domains, such as education, psychology and engineering. One of the earliest platforms for educators to discuss learning and teaching in computer science was hosted by the Special Interest Group in Computer Science Education (SIGCSE) in 1970 (Pears et al. 2005: 152). Other well-established resources and outlets for educators include the Computer Science Education journal, which was first published in 1988 (Fincher & Petre 2004: 1) and is broadly concerned with all matters relating to Computing Education Research (CER), and the Innovation and Technology in Computer Science Education (ITiCSE) conference series, which originally convened in 1995 (Pears et al. 2005: 152). There are also specialist workshops, such as the Psychology of Programming Interest Group (PPIG) and the workshops of the Empirical Studies of Programmers (ESP), that are exclusively concerned with the learning and teaching of programming. Other regional and international conferences, such as the Australian Computing Education Conference (ACE) and the International Computing Education Research Workshop (ICER), also represent wider interests of computing education.

The subject of Computer Science Education research (CSEd), which is sometimes referred to as Computing Education Research (CER), addresses widely ranging features of the learning and teaching process. Many authors have attempted to categorise the main CER themes. These include the works of Berglund and colleagues (2006), Valentine (2004), Fincher and Petre (2004), Glass and co-workers (2004) and Kinnunen (2010). Fincher and Petre (2004: 3) identify “ten broad areas that motivate researchers in CS education.” These comprehensively represent the overall body of CSEd research and provide part of the rationale for the subject areas and approaches selected in this research:

1. Student understanding: explores how students learn CS concepts, underpinned by cognitive and neuro-psychological theories of learning considered in Section 2.2.2;
2. Animation, visualisation and simulation: concerns the creation of tools that aid the representation of CS concepts, for example, game-based learning visualises 'intangible' programming concepts and is discussed in Section 2.3.3;
3. Teaching methods: evaluates and discusses specific methods that communicate CS concepts in teaching sessions. These are also considered in Section 2.3.2;
4. Assessment: is concerned with the range of techniques for measuring performance in CS degrees, and is considered in Section 2.3.3;
5. Educational technology: evaluates the impact of tools such as VLEs and smartphones on learning processes. Smartphone and internet usage are considered in Section 2.2.3;
6. Transferring professional practice into the classroom: discusses the extent to which classroom practices reflect industry working expectations and formats. These are discussed in Sections 1.4 and 2.3.1;
7. Incorporating new developments and new technologies: these are distinct from the previous category of research (6) and are specifically concerned with the implementation of curriculum modifications. The challenges of introducing such changes are considered in Section 2.3.2;
8. Transferring from campus-based teaching to distance education: designs teaching and learning content to be delivered online. These measures are associated with the changing expectations and working habits that are discussed in Sections 2.2.3 and 2.3.2;
9. Recruitment and retention: identifies the motivations for enrolment and causes of withdrawal, in addition to evaluating the effectiveness of interventions targeting these areas. This subject is addressed in Sections 1.3 and 2.3;
10. Construction of the discipline: discusses the importance and order of individual topics in CS curriculums. Introductory programming curriculums are briefly referenced in Sections 1.4 and 2.3.1.

Although research is conducted into most syllabus areas of CS degrees, the teaching and learning of computer programming has attracted considerable attention (Section 1.2). Sheard and co-researchers (2009: 95) undertook a systematic review of the CSEd proceedings of six renowned conferences, finding that 349 of 979 publications between 2005 and 2008 were concerned with programming. Ihantola and colleagues (2015) also found that the volume of programming research published in 2015 was significantly greater than in 2005.

Despite the growing body of CSEd research, there are issues that undermine confidence in the findings reported, and consequently, their impact (Porter et al. 2013a). CSEd researchers often lack training in educational research resulting in poor understanding of underpinning theory and alignment with existing research (Fincher & Petre 2004; Robins 2015). There are few clearly established ways of conducting and validating research in the field (Doane 2013: 1) and CSEd is not always consistent with general approaches for education research (Almstrum et al. 2005). Researchers also struggle to shape a culture for CSEd research due to the differing standards and publication requirements of conferences and journals (Fincher & Petre 2004). The challenges faced by the CSEd research community are considered in greater detail below.

3.1.1 The lack of specific methodology

Although much CSEd research has been conducted since Computer Science courses became established in the 1960s, there is still little consensus as to which methods and approaches represent best practice (Doane 2013: 1; Robins 2015). CER and CSEd research tend to draw on "more established traditions" such as qualitative measures used in cognitive psychology and education, as well as the quantitative techniques used in scientific research (Malmi et al. 2014: 27). Similarly to indications in Section 2.3.2 that CS educators continue to teach using familiar approaches, CS researchers also tend to adopt conservative rather than innovative methods, regardless of the fact that other approaches may be better suited for investigations (Robins 2015).

Few published CER papers and articles clearly describe the methodology adopted (Pears et al. 2005: 154; Randolph 2007; Sheard et al. 2009). Randolph (2007) reviewed a selection of computing education publications that did describe methodology and research processes but concluded that most were neither adequate nor rigorous. Doane (2013: 1) observed that "CER papers typically rely on unvalidated instruments, if any, to assess student attitudes toward computing or toward a particular class or tool, and as such, offer little to no generalisability."

3.1.2 The lack of research training

CS researchers and educators have backgrounds in many different fields, including education, computer science, technology and engineering (Fincher & Petre 2004: 2). Robins (2015: 115) found that most CSEd researchers are "CS teachers drawn to educational research out of the frustration of witnessing the difficulties experienced by our students." However, their CS qualifications may not have exposed them to educational theories, research methods or statistics (Pears et al. 2005: 153).

Robins (2015) encourages CS educators to acquire a proper understanding of social science research methods and a working knowledge of statistical tests and analysis. Doane (2013: 1) also agrees that improved development and training of researchers is likely to improve the quality of research publications. Case studies of explicitly teaching research methods from a CER perspective have been largely beneficial (Berglund et al. 2007), but smaller CS departments have less time and fewer resources to invest in training CSEd researchers (Cooper et al. 2016).

3.1.3 Underpinning CSEd research with established general education research

At a SIGCSE workshop in 2005, a panel of CSEd and CER researchers agreed that, too often, CSEd research disregards the substantial body of prior research in education, cognitive science and learning science (Almstrum et al. 2005). Few CER publications refer to existing or proposed learning theories (Doane 2013; Ihantola et al. 2015: 44).

However, in their review of 308 journal articles published between 2005 and 2011, Malmi and co-researchers (2014) discovered that 157 papers (51%) referred to at least one model, theory or framework. The learning philosophy of constructivism (and derivatives) was frequently referenced to underpin research investigating how students learn and create programming knowledge (Malmi et al. 2014). The fact that this study was inspired by Ben-Ari's 1998 review of constructivism in CS education testifies that the relevance of constructivist theory has for some time been accepted by CS researchers. Fincher and Petre (2004: 13) contrast differing epistemological views of social scientists and natural scientists. Theory in natural science fields generally refers to cause-effect relationships and predictability. Theory eventually and objectively becomes 'truth' when results are replicated across multiple circumstances. Conversely, social science theories concentrate on the reasons for effects, "the causes for which may not be determined, or determinable" (Fincher & Petre 2004: 13). The authors record that much published CER research recognises and describes student behaviour in teaching sessions, although such observations require greater support from educational theory to elevate apparent 'recognition of phenomena' to the status of 'evidence' (Fincher & Petre 2004: 15).

However, Malmi and colleagues (2014: 28) state that general educational theories or even theories from neighbouring fields, such as engineering education research, are not fully able to explain phenomena, "for example, how students understand programming concepts." The authors, therefore, suggest that the CSEd and CER communities are best qualified to determine appropriate research methods for their discipline (Lister 2005a; Malmi et al. 2014: 28). Nevertheless, Malmi and co-researchers (2014) expect such understanding to be constructed and contextualised with reference to established or widely accepted theories where appropriate.

Pears and colleagues (2005: 155 - 156) recommend that CS researchers should study theories such as constructivism, situated learning and social learning theory. Knowledge of these theories and others from psychology and education is important for helping CS educator-researchers to develop a broader research perspective and to enhance confidence in research findings.

3.2 Methods of Computer Science Education research

This section reviews published CSEd work to inform method selection for the research conducted here. The review is structured to address four areas of interest: (1) the type of research; (2) methods of data collection; (3) the size or sample representation of studies and sample sizes; and (4) methods for data analysis.

3.2.1 Type of research

As previously established, CSEd research draws on a wide variety of qualitative and quantitative approaches. Krathwohl (1993: 740) defines the aim of qualitative research, as "describ[ing] phenomena in words", and quantitative research as "describ[ing] phenomena in numbers and measures." The two approaches are used to answer distinct research questions. Quantitative research tends to answer questions of 'how many?' or 'to what extent?' and phenomena are related, whereas qualitative research attempts to understand reasons 'why' and mechanisms 'how' observable phenomena occur or relate. Examples of such phenomena include not progressing with studies or collaboratively solving problems. Malmi (2010: 23) suggests that examples of qualitative CSEd research questions might be along the lines of 'How do students understand concept/process X?' or 'What are the reasons behind high dropout rates?' Corresponding examples given for quantitative research include 'Does method X have a positive effect on students' learning results?' or 'What factors have an influence on students' success in CS1?'

CSEd research publications also adopt 'mixed methods' approaches. Johnson and Onwuegbuzie (2004: 17) define mixed methods research as the "combination of quantitative and qualitative research techniques, methods, approaches, concepts or language into a single study".

Malmi (2010) also records that multi-institutional and multi-national (MIMN) and literature surveys are particular types of CSEd research. Sheard and co-researchers (2009: 97) discovered that one quarter of their reviewed studies conducted research across multiple institutions.

Sheard and co-workers (2009: 97) systematically reviewed 164 research papers, finding more quantitative studies (37%) than qualitative studies (21%). However, the 'mixed methods' approach was adopted more frequently than either approach alone (42%). Most of the mixed methods studies were predominately either quantitative or qualitative. For example, questionnaire-based studies were analysed predominantly by statistical tests but often quoted responses from open-ended questions. Conversely, studies conducting qualitative analysis of interview data also reported response frequencies and other numerical summaries (Sheard et al. 2009).

Sheard and colleagues (2009: 97) also observed that certain conferences and publications favoured a particular research approach. The ACM International Computing Education Research (ICER) featured more qualitative than quantitative publications. However, the ACM Innovation and Technology in Computer Science Education (ITICSE) and SIGCSE conferences were found to publish more quantitative papers than qualitative papers. Regional conferences published more papers using the mixed methods approach than adopting single method approaches.

3.2.2 Methods of data collection

Sheard and co-researchers (2009: 97) found that most CS studies (72%) incorporated assessment grades (assignment and exam marks) as a measure of programming ability or programming performance. Researchers also design exams and tests specifically to measure problem solving techniques and to identify mental models (Dehnadi & Bornat 2006; Radermacher et al. 2012). These observations may be correlated with potentially predictive attributes to determine whether these may have value as indicators of learning process and progress. Although educators usually have access to course data (e.g. attendance records, course grades) and enrolment data (e.g. previous qualifications), Sheard and colleagues (2009: 98) found that only 2% of papers analysed supporting and contextual student data (other than assessment grades) from a central database.

CSEd researchers commonly distribute questionnaires to survey student preferences and perceptions, as well as to discover prior education experience and programming knowledge (see also Section 2.2.2). Digital and online questionnaires are useful for their capacity to automatically store and format data, thereby reducing the time required to transcribe paper questionnaires before analysis. Email distribution of digital questionnaires may also be automated to facilitate multi-institutional studies (Bennedsen & Caspersen 2007). However, in a meta-analysis of 35 education studies, Shih and Fan (2009) found that the average response rate of email surveys was 20% less than that for paper surveys.

Questionnaires can be adapted for either qualitative or quantitative research. Qualitative researchers tend to ask ‘open-ended questions’ which require constructed responses (Malmi 2010: 23), whereas quantitative researchers usually ask ‘closed questions’ necessitating selection from a limited choice, such as represented by categorical answers or ordinal-type Likert scales.

When the reasons for quantitative patterns and trends are not immediately apparent from numerical studies, interviews provide useful qualitative opportunities for educators and researchers to discuss these with students (Goldman 2014; Pears et al. 2005: 156). For example, researchers have interviewed students to understand reasons for withdrawing from or failing CS courses (Kinnunen & Malmi 2006), and to identify any difficulties experienced in learning processes (Boustedt et al. 2007). More specifically, interview techniques have been applied to determine the skills required to learn programming (Eckerdal & Berglund 2005). Pears and colleagues (2008) have also interviewed educators to capture their perspectives on teaching computer science.

Researchers commonly use ‘semi-structured’ interviews (Pears et al. 2005) to pilot questionnaires to establish structure, expectations and key questions, before finalising designs for more extensive surveys. Questions may also be presented in advance of interviews to allow participants time to consider responses (Cohen et al. 2011: 441). In contrast to pre-determined questions, the semi-structured format allows discussion and the possibility of probing further. With appropriate consent and ethical approvals, interviews are commonly recorded and transcribed to prevent data loss. Interviewers may also choose to take direct notes of dialogue and interactions.

CS educators may also make notes of in-class observations during or after teaching sessions. Whilst such notes may be subjective (as written from the educator’s point of view), the data can be used to evaluate the effectiveness of teaching interventions if it is analysed in the context of learning theories (Fincher & Petre 2004: 15), and thereby used to inform course modifications. Fisher and co-workers’ 1997 study “*Undergraduate women in computer science: Experience, motivation and culture*” is commonly cited as an exemplary ethnographic study (Pears et al. 2005: 156). Pears and co-researchers (2008) stress the value of describing the diverse experiences of students and teachers to properly represent the broad range of contexts in which CS is taught (Section 2.3.1).

Sheard and co-workers (2009) found that only 5% of a sample of published research reported use of video and audio means for recording student interactions with peers and tutors. Reasons for the low uptake of recorded data collection may be partly due to difficulties in obtaining ethics committee approvals and the informed consent of students. Educators may simply not have sufficient time or resources to prepare environments or to transcribe and analyse recordings (see Section 2.3.2).

3.2.3 The size, extent and representativeness of studies

A systematic literature review by Ihantola and colleagues (2015: 41) found that "the majority of studies are conducted within a single institution and a single course." Although convenient for researchers and educators to record and analyse data from their institutions (Cohen et al. 2011: 155), such studies are unavoidably limited in their representation of wider environmental and contextual circumstances. For example, Dehnadi and Bornat (2006) claimed to have created an accurate test for predicting programming ability. However, further independent studies of the test in other institutions were unable to replicate Dehnadi and Bornat's findings (Casperson et al. 2007; Ford & Venema 2010; Lung et al. 2008; Radermacher et al. 2012; Wray 2007). Bornat (2014) later retracted the claims of his earlier study.

Multi-institutional and multi-national (MIMN) research design can lead to greater confidence in the authenticity of results and evidence that findings are not influenced by extraneous factors (Fincher & Petre 2004; Ihantola et al. 2015). Mike McCracken designed the first MIMN study with colleagues from the UK, USA, Australia and Europe in 2001 (McCracken et al. 2001). Brown and Altadmri (2014, 2015) extended a Java platform to collect student code from institutions worldwide so that any error messages generated could be analysed. Their first study collected data from approximately 110,000 students (Brown & Altadmri 2014) and their next paper had analysed data from 265,000 students from several hundred institutions worldwide (Brown & Altadmri 2015). Although large sample sizes may strengthen conclusions, such studies require considerable administration and time to coordinate. Work by Boyle and co-researchers (2002) and Pioro (2006) demonstrate that it is possible to validate results by replicating studies across two institutions. Such limited replication may present a more realistic option for many busy academics.

However, research conducted at individual and smaller institutions may also make valid contributions to knowledge. Respondents to a questionnaire survey issued by Doane (2013: 1) noted that SIGCSE conferences tended to concentrate on large university contexts and the teaching models used in those environments. Respondents were also of the opinion that small universities and colleges were underrepresented in CSEd literature. The National Science Foundation & US Department of Education (2013: 4) also stressed the need for educational research to be "theory based and justified by appropriate methods - both quantitative and qualitative; from small 'n' case studies to large 'n' studies."

3.2.4 Types of analysis

Sheard and colleagues (2009: 98) found that phenomenography, grounded theory and content analysis were the three qualitative analysis techniques most widely used by CSEd researchers. Phenomenological research requires that participants are subject to multiple interviews so that their views and experiences are properly represented. Each participant's lived experience is considered to be unique, as is the researcher's interpretation of the data collected (Hegel 1807; Husserl 1927). In contrast, the aim of grounded theory research is to generate a theory. This technique originates from a time when 'positivism' was regarded as the only philosophical grounds for research. However, it may also be applied to generate 'interpretivist' theories. Data is collected and analysed simultaneously, and interview questions are repeatedly revised after each round to refine the theory being developed (Glaser & Strauss 1967). Content analysis, on the other hand, is typically used to study documents and media artefacts (Krippendorff 2004). Employing content analysis to analyse data qualitatively involves interpreting meaning and implications from the content of text data (Hsieh & Shannon 2005). Whilst there are similarities between qualitative content analysis and thematic analysis (Vaismoradi et al. 2013), content analysis is frequently used as means for the quantitative analysis of qualitative data. For example, the technique may be used to infer meaning from word or phrase frequencies in documents and transcripts (Sheard et al. 2009).

Sheard and colleagues (2009: 98) found that fewer qualitative research papers specified or clearly described methodology than quantitative research papers. The authors found that nearly 40% of qualitative papers did not report any methodological details. However, Sheard and co-researchers (2009: 98) also found that 43% of quantitative research papers described data with the help of graphs or tables but did not perform statistical tests. The remaining 57% of quantitative papers did employ statistical analysis techniques, although Sheard and colleagues (2009) did not specifically identify the tests used. The review of literature (Sections 2.1 & 2.2) identified that Pearson's Correlation, Spearman's Rank correlation coefficient and regression analysis were the most commonly used statistical tests for evaluating relationships between variables.

3.3 The research approach

This section is concerned with the approaches adopted for the research reported in this thesis. Three subsections are concerned with: (1) summarising the key concerns of research, (2) evaluating the appropriateness of mixed methods in this context; and (3) introducing the individual studies conducted in support of the research focus areas.

3.3.1 The overall focus for research

This subsection summarises the specific interests of this research (informed by unique opportunities identified from Section 2.4) and discusses the most appropriate methods to best answer the associated research questions.

(1) Exploring relationships between post-entry grade performances, application data and previous qualifications

Section 2.4 identified an opportunity to investigate differences in programming performance between BTEC and A-level students and Section 3.2.2 identified that student data held on university databases is rarely analysed (Sheard et al. 2009: 98).

Quantitative and statistical analysis can be used to evaluate the indicative value of previous qualification grades and other student data. Non-numerical demographic information may be assigned to categories to enable programming grades to be compared across gender and socio-economic groups. Correlational tests can determine the strength of association between variables but cannot infer or prove causation (Whitley 1996: 404). Correlational analyses do not account or control extraneous variables in the same way that experimental designs do. However, experimental designs are often not suitable for educational research and for explorations involving historical data, such as previous qualification grades, UCAS points and L4 programming grades.

Where quantitative analysis is useful for revealing patterns in student populations, qualitative analysis of interview and in situ observation data may provide a deeper insight for behaviours that determine the extent to which students engage with and make progress in learning environments.

(2) Exploring the impact of student engagement

Numerical measurements of engagement such as attendance and L4 course grades may be subjected to correlation analyses with L4 programming grades to evaluate their potential usefulness as predictors or early indicators of a need for interventions. However, evaluating the influence of study motives and learning approaches is complex as these are highly variable phenomena (see Section 2.2). By combining both quantitative analysis of questionnaire responses that capture a measurement of motives and engagement metrics at one point in time, and qualitative analysis of interview data to learn more about underlying goals and motives, it is possible to simultaneously establish whether correlations exist and to suggest underlying mechanisms for such associations.

Section 2.4 identified that the video-recording data collection method could meet the need for detailed exploration of collaborative behaviours and tacit interactions between students in teaching sessions. This method would allow researchers to record a broader range of contextual information (such as participant's environment, questions, speech and the nature of tasks or other activity) and to revisit this as necessary to more fully understand student interactions. However, discomfort with being recorded may discourage student participation with the investigation. Sample sizes for detailed qualitative research approaches are often smaller than for quantitative research. This is because each sample is often extremely information-rich. Analysis therefore aims to establish meaning and explanation for trends revealed by quantitative analysis. Such qualitative analysis also provides valuable case-studies that highlight individual diversity and prevent over-generalisation. Additional non-video recorded observations of similar behaviours are useful for further validation, without time-consuming recording and analysis procedures.

(3) Evaluating teaching interventions and curriculum modifications

Section 2.4 identified the opportunity to better facilitate individual student development through formative opportunities, as well as teaching methods which communicate the problem-solving process. However, as educators often have little opportunity to make major modifications to courses (Section 2.3.2), this research is mainly concerned with evaluating the impact of making incremental changes to an introductory programming course.

The impact of assessment modifications and teaching methods can be measured quantitatively, through analysis of grade, attendance and pass rate metrics. However, a multitude of factors can influence student performance in assessments and affect their attendance (Section 2.2). Although these extraneous institutional and individual influences on findings have been explored through quantitative meta-analysis of hundreds of publications (Freeman et al. 2014) and analysing data from many institutions worldwide (Porter et al. 2016), much remains to be done regarding the replication of studies and substantiation of findings. Studies reported here also adopt a mixed methods approach in order that attempts may be made to qualitatively suggest behaviours and the mechanisms of other influences in patterns revealed by quantitative analysis.

3.3.2 Mixed methods

As stated in Section 3.2.1, the mixed methods approach was adopted more frequently than the individual quantitative and qualitative approaches alone in CSEd research (Ahadi & Lister 2013; Sheard et al. 2009; Teague 2011). The approach combines methods from both quantitative and qualitative research approaches as a means for validating findings and to address research questions in greater depth, such as by offering underlying reasons for quantitative relationships (Johnson & Onwuegbuzie 2004). In the past, the approach was heavily criticised due to the perceived incompatibility between positivist and interpretivist philosophies that respectively underpin the quantitative and qualitative research approaches (Johnson & Onwuegbuzie 2004).

Some researchers still maintain the view that the quantitative and qualitative research approaches are opposed epistemologically, as they have an exclusive view on data interpretation. However, proponents of the technique argue that the two perspectives can be complementary rather than contradictory (Cresswell & Plano Clark 2011: 5). Reams and Twale (2008: 133) describe the mixed methods approach as "necessary to uncover information and perspective, increase corroboration of data and render less biased and more accurate conclusions."

The mixed methods approach is philosophically underpinned by 'pragmatism'; a view that acknowledges the 'less than perfect' and complicated nature of conducting research in the real world. Pragmatism views the world as both objective and socially constructed (Johnson & Onwuegbuzie 2004). Pragmatism (and the mixed methods approach) encourages researchers to select methods appropriate for answering the research question, rather than their own preferences or methods traditionally associated with the subject or nature of inquiry (Cohen et al. 2011; Fincher & Petre 2004; Onwuegbuzie & Leech 2005: 377; Phillips & Gilding 2003).

Pragmatism still values rigour and process; it is not an 'anything goes' approach. Mixed methods research must still choose meaningful research questions and answer them with appropriate evidence (Denscombe 2008). Bryman (2007: 8) recommends the selection of a mixture of methods when the research question cannot be answered through an exclusive quantitative or qualitative approach.

Teddlie and Tashakkori (2009: 26, 142, 151) describe different designs for conducting mixed methods research:

- ‘parallel mixed designs’ (otherwise known as ‘concurrent designs’) allow both qualitative and quantitative methods to be conducted simultaneously to answer the same research question;
- ‘sequential mixed designs’ alternate between qualitative and quantitative approaches;
- ‘quasi-mixed designs’ collect qualitative data and quantitative data together but are used to answer separate research questions;
- ‘conversion mixed designs’ convert qualitative data to quantitative data and vice versa.
- ‘multilevel mixed designs’ apply different research approaches in the same study but at different levels (levels refer to the numbers of participants involved – often researchers will start with a small sample size and scale up); and
- ‘fully integrated mixed designs’ apply both research approaches at all levels of the study.

The areas of investigation described in Section 3.3.1 require both quantitative and qualitative analysis techniques to answer different research questions. This research intends to apply quantitative analysis for measuring predictive ability of prior qualifications and course attributes, and qualitative analysis for measuring the impact of teaching interventions and investigating underlying patterns of learning behaviour, thus suggesting that the ‘quasi mixed design’ is most appropriate for the research circumstances reported here. This research will also adopt ‘conversion’ (quantising) as mentioned in Section 3.3.1, for categories that do not inherently have numerical data for the purposes of comparison with grade performance.

3.3.3 Study outlines

This section introduces the studies reported in this thesis and maps these to research areas and the corresponding chapters.

<u>Chapter 4: Exploration of student application data</u>

The studies reported in Chapter 4 aim to fulfil Objective 3 by identifying potential indicators of learning success amongst three academic years of student application data and previous qualification grades. Correlation analyses evaluate the indicative value of student application data (such as UCAS points and age). The L4 programming performance of A-level and BTEC entrants, their gender, geographical area of previous education and nationality are also evaluated with nonparametric boxplots (Tukey 1977). Similar data was collected and analysed from an international partner college for purposes of replication (Fincher & Petre 2004).

International differences regarding pre-undergraduate qualifications meant that specific relationships between UK BTEC and A-level qualifications could not be validated by the partner college dataset. However, additional questionnaire data was collected from one cohort of Bucks students, and interviews were conducted with A-level and BTEC students, to better understand their experiences and to propose possible reasons for trends identified from quantitative analysis (Fincher & Petre 2004: 15).

The research objectives of a number of studies reported here often concerned whether differences existed between the grade performance averages of populations. Examples of hypothesis tests were examining differences between male and female students, between BTEC and A-level entrants, between ethnic groups or Local Education Authority (LEA) regions. For such tests, normality of grade distribution was not assumed and the non-parametric and data-visualisation means of boxplots were applied. Boxplots test for differences in medians of populations and usefully demonstrate data distributions as well as the presence of outliers (Chambers et al. 1983; Tukey 1977). The ggplot2 library (Wickham 2009) of the R language (R Core Team 2013) was used to generate boxplots applied to the analyses reported here.

Where analyses hypothesised that two continuous variables were significantly related, for example UCAS points and programming grade performance, Pearson's correlation was applied. Although parametric assumptions for the normality of grade performance may have been breached for some distributions and Spearman's Rank correlation may have been applicable, Pearson's correlation was nevertheless applied for the reasons that grades are true ratio-scale data. Pearson's is also generally consistent with the approach used by the research sector (Pioro 2006; Stein 2002).

Chapter 5: Exploration of student engagement data

The studies reported in Chapter 5 aim to fulfil Objective 4 by investigating how L4 grades at Bucks and L6 grades at the South Asian Institute for Technology and Mathematics (SAITM) in Sri Lanka correlated with indicators of course engagement. Indicators of course engagement included attendance at both institutions and more detailed indicators from questionnaires concerning engagement behaviours at Bucks.

This study also extends and modifies early work by Mather (2014). Questionnaires were distributed to elicit student perceptions of an L4 introductory programming course to better understand how attitudes and working habits relate to learning progress. The questionnaire adopted a five-item ordinal Likert scale for responses and included a five-question test to evaluate understanding of concepts introduced during the programming course. These variables were subject to correlation and multivariate analyses with other metrics, such as for attendance and grades.

Studies that use audio/visual recordings to explore collaboration amongst programmers (Plonka et al. 2011; Nawahdah & Taji 2015; Mather 2004; Zarb & Hughes 2012) demonstrate a wide range of research approaches and analysis techniques. Thus, authors reporting on the analysis of recordings made of commercial programmers working together (e.g. Plonka et al. 2011; Zarb & Hughes 2012), were mainly concerned with the impact of collaboration on business processes, whereas other researchers were interested in the educational impact of collaboration (Nawahdah & Taji 2015; Mather 2004).

Due to the volume of "rich data" involved, Jewitt (2012) after Snell (2011) cautions researchers against analysis that may become "overly descriptive". Although not using Snell's technique of pairing "systematic quantitative analysis" with "micro-ethnographic qualitative analysis", a similar mixed method approach with limited qualitative sampling is adopted here. Quantitative analysis of the video data measured length of time spent on task as well as word and theme frequencies. Qualitative analysis techniques sought to interpret dialogue and actions for underlying working strategies and motivations for learning. Zarb and colleagues' (2012: 2) two stage analysis process was adopted for efficient interpretation of meaningful video segments. The first stage identifies the most relevant and meaningful video segments for the investigation, and the second stage concentrates on conducting extensive analysis of those segments.

Chapter 6: Evaluation of teaching interventions and course modifications

Chapter 6 aims to fulfil Objective 5 by evaluating the impact of modifications made to Bucks' L4 programming course, following certain recommendations from Chapter 5, through the quantitative analysis of attendance and grade data, supplemented by further qualitative analysis of ethnographic-type observations, interviews and informal conversations. This study discusses the evolution of formative assessment given to successive L4 cohorts at Bucks and includes correlation analysis between formative assessment results and final L4 grades. Summative assessment modifications are also discussed in the light of results from quantitative analysis of attendance and grade data. Course curriculum changes, teaching methods, and remedial interventions are discussed with reference to the qualitative analysis of ethnographic observations and student feedback.

The effectiveness of programming pedagogies that encourage active learning are typically evaluated numerically by their impact on grades and retention (Freeman et al. 2014; Porter et al. 2016), in addition to their impact on numerical measures of engagement such as attendance or perceptions of enjoyment and satisfaction (Fotaris et al. 2015). However, increases to numerical measurements of course engagement do not necessarily reflect improved quality of teaching and learning. For example, students may feel more satisfied with their course if they achieve a high grade. They may attend for factors completely unrelated to teaching style. Even though large MIMN studies that analyse large sample sizes provide confidence in results, these findings are rarely considered in the context of existing educational learning theories and lack explanatory value (Almstrum et al. 2005; Doane 2013).

Even though this research does intend to measure impact on grade and attendance, seeking to understand underlying explanatory reasons for engagement has greater value for informing future teaching methods. Therefore, this research aims to address these concerns by discussing observations, interviews and informal comments in relation to prevalent cognitive and constructivist theories, such as Neo-Piagetian theory (Teague et al. 2013) after Piaget (1952), scaffolding (Robins 2010), cognitive load theory (Sweller et al. 2011), in addition to research concerning attentiveness (Davidson 2011) and cognitive control: distractions and procrastination (Gazzaley & Rosen 2016; Rosen et al. 2011), for evidencing teaching interventions in contemporary contexts (Gordon 2016).

Chapter 4 Exploration of student application data

This chapter is concerned with exploring the usefulness of qualification and individual profile information as predictors of L4 programming progress and as possible early indicators for identifying where instructors may need to support students or otherwise intervene. Section 4.1 investigates the differences between A-level and BTEC computing qualifications and the L4 programming performance of students holding such qualifications. In Section 4.2 the indicative value of other data from university entrance applications is analysed for its usefulness in predicting L4 programming performance. Section 4.3 summarises the findings of Section 4.1 and Section 4.2.

4.1 A-Level and BTEC qualifications

4.1.1 Quantitative analysis of factors associated with programming grade performance

Methods

The quantitative analysis in this section compares L4 programming performance of students who have undertaken previous courses in A-level Computing, A-level IT, BTEC IT or BTEC Games. Level 4 grade data, entry qualification grades and subject data were obtained for academic years 2013/14 (n = 43), 2014/15 (n = 39) and 2015/16 (n = 38). All the students that participated in this study were fully informed of the purpose of the research during classes and in writing (Appendix 1). Although students will have consented to the collection and analysis of their profile data when signing their enrolment form (Appendix 2: Section 8) before starting their undergraduate degree course, individuals were allowed to withdraw from the study. This study was also approved by an internal ethics panel (Appendix 3).

The distribution of L4 programming performance for different qualification categories is represented and analysed using notched boxplots. These make no parametric assumptions (Tukey 1977) and are combined with significance tests for the centrality of distributions. Notches represent 95% confidence intervals (CI) for the median. If notches for two distributions do not overlap the respective medians may be regarded to be significantly different at p=0.05 (Chambers et al. 1983: 62). To aid interpretation of boxplots, it should be noted that the median is represented by the dark horizontal lines centrally bisecting vertically orientated boxes. The notch/waist extends to the 95% confidence limits for the median. The box represents the interquartile range and dashed lines extend to values within a distribution represented by up to 1.5x the interquartile range. Outliers are represented by single points.

This study also aimed to analyse comparable similar data from a partner college, the South Asian Institute for Technology and Mathematics (SAITM) in Sri Lanka. The proposal for this study was submitted to SAITM tutors for scrutiny (Appendix 4) along with the ethical approval granted for the study at Bucks (Appendix 3). The SAITM tutors gave their permission for the study to proceed (Appendix 5) as students had consented to data collection and analysis when signing their enrolment form (Appendix 2: Section 8) and on the basis that findings would be reported anonymously. Bucks' internal ethics panel also extended the permission granted in the original study to collect and analyse data from SAITM (Appendix 6). However, students in Sri Lanka who undertook BTEC HND qualifications were assessed at L4 and L5, in contrast to UK BTEC qualifications, which are assessed at L2 and L3. Whilst direct comparisons with UK data could not be made, BTEC qualifications were tested for their potential to predict L6 programming grades for both UK and Sri Lanka based students.

The L4 programming performance of BTEC and A-level students (Bucks)

Initial analysis assigned sampled students to four categories (A-level Computing, A-level ICT, BTEC IT, and BTEC Games). Students who possessed more than one previous computing qualification were allocated to the qualification category corresponding with their highest grade. Students with low grades or no grade at all (U) were removed as outliers and because there is little reason to suspect that an unsuccessful attempt to pass or low achievement will be of any advantage (Robins 2010: 38).

Further analysis included students who took alternative A-level subjects to Computing or IT subjects, and those who studied alternative BTEC subjects to IT or Games. Students who did not pass computing related qualifications were also placed in respective non-computing A-level or BTEC groups, providing they passed at least one non-computing qualification. Students possessing both A-level and BTEC qualifications were placed in the category representing their highest award. Students who applied directly to the university were also assigned to a separate category for analysis.

Figure 4.1 indicates that the category for A-level Computing students demonstrated a smaller grade range and a higher median value than the A-level ICT students. Drummond (2009: 112-115) also found that L4 programming grade averages were less for A-level ICT entrants (at 59%) than for A-level Computing entrants (at 61%). She also discovered the distribution of marks for A-level ICT students (s.d. = 13.1%) was greater than that for A-level Computing students (s.d. = 9.3%). Figure 4.1 illustrates that the median scores for both A-level ICT (82%) and A-level Computing (86%) students are greater than those for both BTEC IT (60%) and Games (46%) students.

This notched boxplot analysis also demonstrates that the median Programming Concepts grade for the non-computing A-level entrants (78%) was significantly greater (*p* = 0.05) than that of non-computing BTEC entrants (43%). The seemingly anomalous distribution for A-level ICT, A-level Computing, and BTEC Games is due to the confidence interval for the median being greater than the interquartile range. This commonly occurs with small sample sizes and widely ranging distributions. Despite an unusual appearance, the 95% CI may still be interpreted as it is indicated.

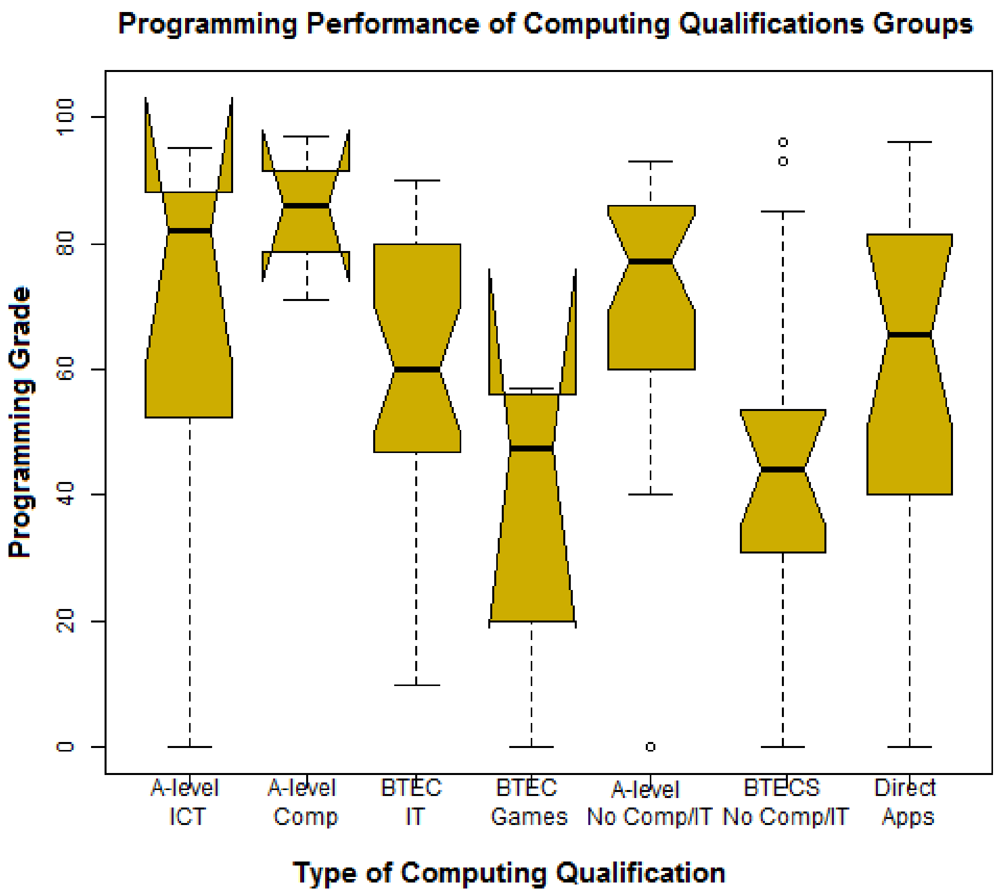


Figure 4.1. Notched boxplot comparisons for L4 programming grade across specific entry qualification categories. Notes: (1) n for categories is: A-level ICT, 7, A-level Computing, 3; BTEC IT, 26; BTEC Games, 4; A-levels without Computing/IT, 30; BTECs without IT/Games, 16 & Direct Application, 18; (2) a grade of 0 typically indicates a non-submission.

More students enrolled on computing related degree courses with BTEC qualifications than with A-level qualifications (Note 1 attached to Figure 4.2). This corroborates the recent increase in undergraduate applicants holding BTEC awards, reported in Section 2.1. There has also been a rise in the number of students taking a combination of BTEC and A-level awards (UCAS 2016c: 16, 34).

Figure 4.2 reveals that the median L4 programming grade for A-level entrants (78%) was significantly greater (at *p*=0.05) than that for the BTEC entrants (50%). This analysis excludes entrants for whom previous education data was unavailable, although it is likely that students admitted through direct applications may have also held A-level or BTEC awards.

On average, those students entering with a combination of BTEC and A-level awards (82%) achieved significantly higher programming grades than those with BTECs alone (50%) (Figure 4.2).

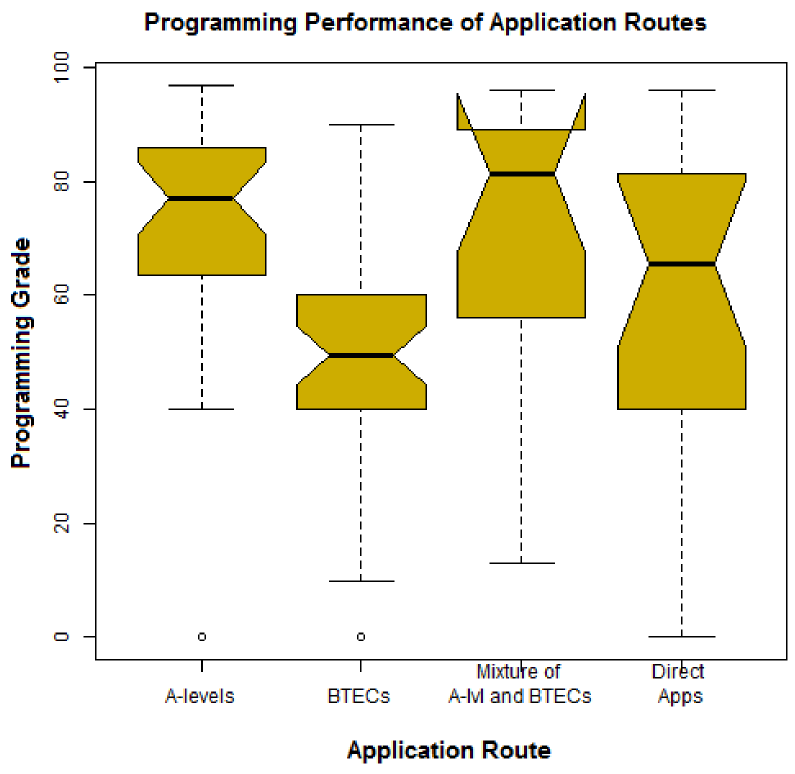


Figure 4.2. Notched boxplot comparisons for programming grade across entry qualification categories. Notes: (1) n for categories is: A-levels, 32; BTEC, 38; A-Levels with BTECs 14 & Direct Application, 20; (2) the anomalous distribution for 'mixture of A-Levels and BTECs' group is caused by the confidence interval for the median extending beyond the interquartile range. This commonly occurs with small samples.

<u>The indicative value of UCAS points (Bucks)</u>

Students holding both BTEC and A-level awards (280) achieved the most UCAS points on average (Figure 4.3). This could be a result of having more qualifications than students who exclusively took BTEC or A-level qualifications. Notched boxplot analysis indicates the median value of this combined group is not significantly different to groups of A-level students and BTEC students (Chambers et al. 1983).

Both the BTEC and the combined A-level and BTEC groups achieved a greater upper quartile than the group of A-level students. This is consistent with findings reported in Section 2.1, that suggest more BTEC students are achieving top grades than A-level students (HEFCE 2015; Reidy 2015).

However, whilst A-level students scored fewest UCAS points on average (180) (Figure 4.3), their median L4 programming grade (78%) was similar to that of the mixed group (82%), and significantly greater than that for BTEC students (50%) (Figure 4.2).

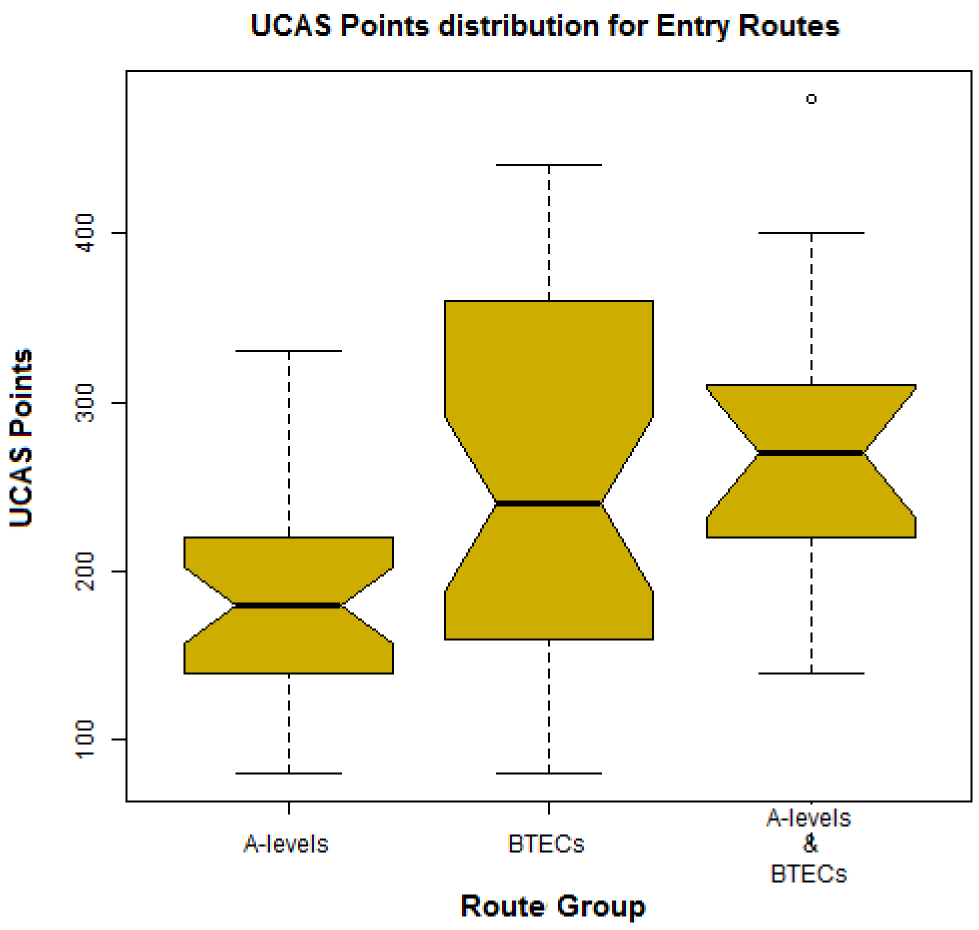


Figure 4.3. Notched boxplot comparisons for UCAS points across entry qualification categories. Note: n for categories is: A-levels, 32; BTEC, 38 & A-Levels with BTECs 14.

Table 4.1 illustrates that there is a stronger relationship between UCAS points and L4 programming grade for BTEC entrants (r = 0.403) than for A-level entrants (r = 0.125). The misalignment between the UCAS points and the L4 programming performance of A-level students (comparison of Figure 4.2 and Figure 4.3) is confirmed by the weak Pearson's correlation (r) in Table 4.1.

Table 4.1. Pearson's Correlation analysis of UCAS Points and L4 Programming Grade.

	UCAS points (all)	UCAS points (A-level)	UCAS points (BTEC)	UCAS points (both BTECs and A-levels)

L4 Programming Grade	r = 0.191
	p = 0.079
	n = 86
		r = 0.125
		p = 0.487
		n = 33
			r = 0.403*
			p = 0.015
			n = 36
				r = 0.376
				p = 0.167
				n = 15

Note: * Correlation is significant at the 0.05 level (2 tailed).

Further comparisons reveal that UCAS points are generally weak indicators of related L4 grades summarises comparative data for six L4 modules on Bachelor Degree courses at Buckinghamshire New University (see Table 4.2). Correlation between Semester 2 Programming Grade and UCAS points follow a similar pattern to correlation between Semester 1 Programming Grade and UCAS points. Whilst A-level students appear to outperform BTEC students in L4 programming, it appears that BTEC qualification grades are more strongly associated with L4 Programming Grade (Semester 1). This trend is investigated further in Section 4.1.2.

Table 4.2. Pearson's Correlation analysis of UCAS points and L4 grades for other modules.

	Semester 1 Computer Architectures Grade	Semester 2 Networking Grade	Semester 1 Digital Technologies Grade	Semester 1 User Experience Grade	Semester 1 Programming Grade	Semester 2 Programming Grade
UCAS Points (all)	r = -0.060 p = 0.600 n = 78	r = 0.103 p = 0.369 n = 78	r = 0.057 p = 0.658 n = 63	r = 0.161 p = 0.208 n = 63	r = 0.191 p = 0.079 n = 86	r = 0.217 p = 0.098 n = 59
UCAS points (A-Level)	r = -0.101 p = 0.595 n = 30	r = 0.038 p = 0.844 n = 30	r = 0.161 p = 0.485 n = 21	r = 0.013 p = 0.485 n = 21	r = 0.125 p = 0.487 n = 33	r = 0.074 p = 0.764 n = 19
UCAS points (BTEC)	r = 0.054 p = 0.772 n = 31	r = 0.097 p = 0.605 n = 31	r = 0.319 p =0.098 n = 28	r = 0.270 p = 0.164 n = 28	r = 0.403* p = 0.015 n = 36	r = 0.366 p = 0.066 n = 26
UCAS points (Mixed)	r = 0.062 p = 0.826 n = 15	r = 0.105 p = 0.710 n = 15	r = -0.144 p = 0.655 n = 12	r = 0.271 p = 0.395 n = 12	r = 0.376 p = 0.167 n = 15	r = 0.612* p = 0.034 n = 12

Notes: (1) * Correlation is significant at the 0.05 level (2 tailed); (2) data is taken from academic years 2013-2015.

The indicative value of equivalent UCAS points (Sri Lanka)

The total value of BTEC points was used as an equivalent measurement in the absence of UCAS points for Sri Lanka data. This variable ('total BTEC points') was created by accumulating the quantised BTEC unit awards. Although the intervals between UK UCAS points (ten percentage units for AS and 20 for A2) are greater than the intervals for Sri Lankan BTEC awards (intervals of one percentage unit), both point systems are proportional to achievement.

Table 4.3 illustrates comparisons between prior qualification grade and undergraduate programming grade for both Sri Lankan and Bucks datasets. The relationship between Sri Lankan BTEC points (L5) and L6 programming grade (r = 0.421, p = 0.008, n = 39) is similar to the relationship between UK BTEC points and L4 programming grade (r = 0.403, p = 0.015, n = 36). Interestingly, these BTEC courses are taught at different levels (Sri Lankan BTEC HND: L4 and L5; whereas UK BTEC: L2 and L3) and are for different subjects (Sri Lankan students studied BTEC Interactive Media, whereas UK students studied BTEC IT or BTEC Games Development).

Table 4.3. Comparisons between prior qualification grades and undergraduate programming grades for Bucks and Sri Lankan students.

	Sri Lanka total BTEC points	Bucks UCAS points	Bucks BTEC UCAS points	Bucks A-level UCAS points	Bucks mixture of BTECs & A-levels
Undergraduate Programming Performance	r = 0.421** p = 0.008 n = 39	r = 0.191 p = 0.079 n = 86	r = 0.403* p = 0.015 n = 36	r = 0.125 p = 0.487 n = 33	r = 0.376 p = 0.167 n = 15

Notes: (1) 'Undergraduate Programming Performance' refers to the L6 Advanced Interactive Programming (AIP) course for Sri Lankan students, and the L4 Programming Concepts course for UK students; (2) ** Correlation is significant at the 0.01 level (2-tailed); (3) * Correlation is significant at the 0.05 level (2-tailed).

Comparisons between UCAS points and L6 Programming grade (Bucks)

Bucks L6 Advanced Interactive Programming (AIP) grade was included in correlations with UCAS points. Table 4.4 suggests that BTEC grades are stronger indicators (r = 0.790) of L6 programming grade than A-level grades (r = 0.293), similar to those patterns found amongst L4 data (Table 4.1), respectively, r=0.403 for BTEC grades and r=0. 125 for A-level grades. Table 4.4 demonstrates an exceptionally strong negative relationship (r = -0.971) between those who applied with both BTEC and A-level qualifications and their L6 programming grade. This is in marked contrast to other relationships in Table 4.4, all affording positive values for the r-coefficient. This anomalous relationship for the BTEC/A-level category corresponds to a small number of students and may be the product of factors unrelated to prior qualifications. It therefore merits further investigation.

Another notable difference between L4 and L6 relationships is the moderately strong and significant relationship (r = 0.404) between the UCAS points variable containing all qualification types (A-levels, BTECs and combination of A-levels and BTECs) and L6 programming grade (Table 4.4), which was not apparent for L4 comparisons (r = 0.191) (Table 4.1). However, the numbers enrolling for L6 programming were considerably fewer than those for L4 programming. There are two probable reasons for this: (1) those who failed Level 4 programming would not have progressed to L6; and (2) L4 programming was compulsory, but because L6 programming was optional it was more likely to attract only those most motivated to study the topic further. However, those who do pass L4 programming also tend to pass L6 programming, as indicated by the moderately strong and highly significant correlation between L4 programming grade and L6 programming grade (r = 0.418, p = 0.005, n = 43).

Table 4.4. Correlations between UCAS points and L4 and L6 Programming Grade for Bucks students.

	UCAS points of Bucks A-level students	UCAS points of Bucks BTEC students	UCAS points of all Bucks students	UCAS points of Bucks students with both BTECs & A-levels

L4 Programming grade	r = 0.125
	p = 0.487
	n = 33
		r = 0.403*
		p = 0.015
		n = 36
			r = 0.191
			p = 0.079
			n = 86
				r = 0.376
				p = 0.167
				n = 15

L6 Programming grade	r = 0.293
	p = 0.289
	n = 15
		r = 0.790**
		p = 0.001
		n = 14
			r = 0.404*
			p = 0.020
			n = 33
				r = -0.971*
				p = 0.029
				n = 4

Notes: (1) ** Correlation is significant at the 0.01 level (2-tailed); (2) * Correlation is significant at the 0.05 level (2-tailed).

Table 4.5 reveals that the L6 programming grade averages for BTEC (52%) and A-level (55%) entrants are similar to the median values for L4 programming students (Figure 4.2). Although average L6 programming grades fall within a narrower range, A-level students achieved both higher L4 and L6 programming grades than BTEC students.

Table 4.5. Average Grades of L6 Bucks students.

	Bucks BTEC students	Bucks A-level students	Bucks mixture of BTECs & A-levels
L6 Programming average grade	52	55	59

Note: n for categories is A-level, 15; BTEC, 14; Mixture of BTECs and A-levels, 4.

Table 4.6 illustrates moderately strong correlations between total BTEC points and the grades for all L6 courses (r = 0.421; 0.430; 0.471), with the notable exception of the final year Project (r = 0.280). This exception may be due to the different delivery and assessment formats for the Project. Unlike the other L6 courses, which are delivered as weekly lectures with prescribed reading material and assignments, the Project requires that students work independently with occasional guidance from a project supervisor.

Table 4.6. Comparisons between the total BTEC grades and grades for all Sri Lankan courses.

	L6 Programming Grade	L6 Software Engineering Grade	L6 Graphical Application Development Grade	L6 Project Grade

Total BTEC Points	r = 0.421**
	p = 0.008
	n = 39
		r = 0.430**
		p = 0.006
		n = 40
			r = 0.471**
			p = 0.002
			n = 39
				r = 0.280
				p = 0.081
				n = 40

Note: ** Correlation is significant at the 0.01 level.

Despite the weak relationship between Project Grade and total BTEC points, Table 4.7 indicates that average grades for each of the four Sri Lankan courses lie within ten points of each other (varying between 53% and 63%). It was not possible to calculate equivalent average scores for L6 Bucks courses such as Graphical Application Development (which is not taught at Bucks in the UK), or Software Engineering (which is only available to students enrolled on the Software Engineering degree programme). However, the average grades for L6 Programming and L6 Project are consistent with each other and the Sri Lankan courses.

Table 4.7. Average grades for Sri Lankan and Bucks L6 courses.

	L6 Programming	L6 Software Engineering	L6 Graphical Application Development	L6 Project
Sri Lankan students	63	53	62	57
Bucks students	56	NA	NA	58

Notes: (1) n for SL categories is SL L6 Programming grade, 39; L6 SL Software Engineering, 40; L6 SL GAD, 39; and SL Project, 40; (2) n for Bucks categories is L6 Programming grade, 42; L6 Project, 91.

Table 4.8 extends upon earlier comparisons performed in Table 4.4 to include correlations between UCAS points and L6 Project grade. This illustrates that UCAS points are, in general, more strongly associated with the L6 Programming grade than the L6 Project grade. However, BTEC qualifications are more accurate predictors than A-level qualifications for both L6 programming grade (r = 0.790 versus 0.293) and L6 Project grade (r = 0.491 versus -0.241) (Table 4.8).

Section 2.1 reported that UCAS points were regarded to be weak predictors of final degree classification (Boyle et al. 2002; Drummond 2009; Gill & Rodeiro 2014). However, the comparisons performed in this study appear to indicate that UCAS points of BTEC students do predict L6 Programming Grade with a strong degree of accuracy.

Table 4.8. Comparisons between the UCAS points and L6 Bucks course grades.

	Bucks A-level UCAS	Bucks BTEC UCAS	Bucks UCAS (all)	Bucks mixture of BTECs & A-levels

L6 Programming grade	r = 0.293
	p = 0.289
	n = 15
		r = 0.790**
		p = 0.001
		n = 14
			r = 0.404*
			p = 0.020
			n = 33
				r = -0.971*
				p = 0.029
				n = 4

L6 Project grade	r = -0.241
	p = 0.293
	n = 21
		r = 0.491
		p = 0.054
		n = 16
			r = 0.053
			p = 0.715
			n = 49
				r = 0.209
				p = 0.563
				n = 10

<u>The indicative value of BTEC unit grades (Sri Lanka)</u>

Prior to enrolling on L6 courses, students undertook a wide range of units (including Art & Design, Animation, Video Editing & Sound Production, Website Design and Computer Programming) for their BTEC HND qualification. Appendix 7 presents the complete BTEC Interactive Media specification and Appendix 8 identifies the 16 units that most students completed prior to enrolling.

Despite the moderately strong and highly significant relationship between total BTEC points and L6 programming grade (Table 4.6), Pearson's Correlation analysis revealed only a weak and non-significant relationship between the Computer Programming BTEC unit grade and L6 programming grade (r = 0.207) (Table 4.9). Analysis also revealed that the BTEC Computer Fundamentals (r = 0.399) and Website Creation and Management (r = 0.409) units were more strongly related with L6 programming grade than the BTEC Computer Programming unit.

Attempts to locate information regarding the content and assessment of the Computer Programming unit were unsuccessful. Limited information was available online and SAITM were also unable to provide further insight for the qualification.

Table 4.9. Pearson's Correlation analysis between L6 Programming Grade and related BTEC units.

	BTEC Computer Programming	BTEC Computing Fundamentals	BTEC Web Creation and Management	BTEC Computer Interface Design Principles

L6 Programming Grade	r = 0.207
	p = 0.206
	n = 39
		r = 0.399*
		p = 0.014
		n = 37
			r = 0.409*
			p = 0.011
			n = 38
				r = 0.310
				p = 0.062
				n = 37

Note: * Correlation is significant at the 0.05 level (2-tailed).

The distribution of awards for the Computer Programming BTEC unit was consistent with other related BTEC units (Table 4.10). This consistency implies that BTEC grade allocation does not appear to contribute towards the weak relationship between BTEC Computer Programming unit grades and L6 programming grades.

Table 4.10. The distribution of awards for computing related BTEC units.

	BTEC Computer Programming	BTEC Computing Fundamentals	BTEC Web Creation and Management	BTEC Computer Interface Design Principles
Distinction	4	4	6	5
Merit	8	9	14	15
Pass	29	26	20	19

There is not sufficient evidence to conclude why Computer Programming BTEC unit grade has a weak relationship with L6 programming grade. However, the moderately strong correlation between total BTEC points and L6 programming grade may indicate that consistent performance across all BTEC units is more indicative than performance in one programming unit. UK BTEC applicants were not required to include full transcripts listing individual unit awards; however, Section 4.1.2 explores the extent to which programming is taught in BTEC and A-level qualifications.

Further Pearson's Correlation analysis revealed weak and non-significant relationships between Computer Programming BTEC unit grade and L6 Software Engineering grade (r = 0.172), L6 Graphical Application Development grade (0.250) and L6 Project grade (0.144) (Table 4.11). On the other hand, grades for BTEC Computing Fundamentals (0.399) and BTEC Web Creation and Management BTEC (0.409) units were moderately strongly correlated with L6 Programming grade. Grades for the two BTEC modules, Computing Fundamentals, and, Web Creation and Management, were found to demonstrate similar strengths of correlation with the grades for L6 Software Engineering (respective r values of 0.540 and 0.243) and Graphical Application Development (0.540 and 0.334).

Table 4.11. Comparison between computing related BTEC units and L6 courses.

	BTEC Computer Programming	BTEC Computing Fundamentals	BTEC Web Creation and Management	BTEC Computer Interface Design Principles

L6 Programming Grade	r = 0.207
	p = 0.206
	n = 39
		r = 0.399*
		p = 0.014
		n = 37
			r = 0.409*
			p = 0.011
			n = 38
				r = 0.310
				p = 0.062
				n = 37

L6 Software Engineering Grade	r = 0.172
	p = 0.289
	n = 40
		r = 0.540**
		p = 0.001
		n = 38
			r = 0.243
			p = 0.136
			n = 39
				r = 0.384*
				p = 0.017
				n = 38

L6 Graphical Application Development Grade	r = 0.250
	p = 0.125
	n = 39
		r = 0.540**
		p = 0.001
		n = 37
			r = 0.334*
			p = 0.041
			n = 38
				r = 0.193
				p = 0.252
				n = 37

L6 Project Grade	r = 0.144
	p = 0.377
	n = 40
		r = 0.258
		p = 0.117
		n = 38
			r = 0.445**
			p = 0.004
			n = 39
				r = 0.246
				p = 0.137
				n = 38

Notes: (1) * Correlation is significant at the 0.05 level (2-tailed); (2) ** Correlation is significant at the 0.01 level (2-tailed).

Further exploration revealed that other creative BTEC units (such as 3D Computer Modelling and Animation) and managerial BTEC units (such as Professional Practice in Art & Design, and Marketing using Interactive Media) were more strongly and significantly correlated with L6 programming grades (with r values between 0.397 and 0.506) than with the Computer Programming unit (0.207) (Table 4.12). Further investigation is required to determine whether there are specific creative and managerial skills or other factors that influence programming ability.

Table 4.12. Comparison between creative and managerial BTEC units and L6 courses.

	BTEC Professional Practice in Art & Design	BTEC Interactive Media Principles	BTEC 3D Computer Modelling and Animation	BTEC Sound Production and Editing Using Interactive Media	BTEC Marketing Using Interactive Media	BTEC Digital and Video Production and Editing
Programming Grade	r = 0.506** p = 0.001 n = 37	r = 0.250 p = 0.135 n = 37	r = 0.460** p = 0.003 n = 40	r = 0.331* p = 0.039 n = 39	r = 0.397* p = 0.015 n = 37	r = 0.339* p = 0.032 n = 40
Software Engineering Grade	r = 0.297 p = 0.070 n = 38	r = 0.379* p = 0.019 n = 38	r = 0.400** p = 0.010 n = 41	r = 0.337* p = 0.033 n = 40	r = 0.489** p = 0.002 n = 38	r = 0.412** p = 0.007 n = 41
Graphical Application Development Grade	r = 0.390* p = 0.017 n = 37	r = 0.249 p = 0.137 n = 37	r = 0.400* p = 0.011 n = 40	r = 0.423** p = 0.007 n = 39	r = 0.470** p = 0.003 n = 37	r = 0.410** p = 0.009 n = 40
Project Grade	r = 0.373* p = 0.021 n = 38	r = 0.327* p = 0.045 n = 38	r = 0.304 p = 0.053 n = 41	r = 0.156 p = 0.338 n = 40	r = 0.294 p = 0.073 n = 38	r = 0.307* p = 0.051 n = 41

Notes: (1) * Correlation is significant at the 0.05 level (2-tailed); (2) ** Correlation is significant at the 0.01 level (2-tailed).

4.1.2 Qualitative analysis of entry qualifications

Methods

The aim of the qualitative analysis in this section is to learn more about the degree to which BTEC IT, BTEC Games Development, A-level ICT and A-level Computing courses prepared students for L4 programming studies. The two research questions posed were:

- To what extent was programming taught in these previous qualifications?
- How effective were these qualifications in preparing students for L4 programming?

This would provide some insight as to why A-level students outperformed BTEC students and, more specifically, suggest why A-level Computing students achieved higher grades than A-level ICT.

Students who had enrolled with these entrance qualifications were invited to participate in an interview by email (the Information Sheet is presented in Appendix 9). Those who agreed to participate were then sent questions prior to being interviewed. Students were asked to sign a Consent Form (Appendix 10) to confirm that their participation was voluntary and properly informed. The Information sheet, Consent Form, and proposal outlining the methods of the study were submitted for scrutiny and approval by the University Ethics Panel (Appendix 11). Students also agreed that interviews could be recorded and that their anonymised data could be analysed.

Points helpful for explaining trends were noted during interviews to assist with processing interview transcripts. Key findings from interviews, along with the outcomes from the prior knowledge questionnaires, were discussed in the light of other curriculum documents and used to create case studies for each entry qualification. A case study can be defined as a "strategy for doing research which involves empirical investigation of a particular contemporary phenomenon within its real-life context using multiple sources of evidence" (Robson 2002: 178). Case studies are the "preferred strategy when 'how' or 'why' questions are being posed" (Yin 2009: 9).

BTEC IT

The content taught in BTEC IT courses varied greatly between institutions. Schools and colleges work from the same specification, but the 2010 BTEC IT course has 43 units of which only three are core units and the rest are optional units or optional specialist units (Pearson 2010a: 24). The course specification indicates that only three out of 30 units relate to programming (Appendix 12). Interviews revealed that the selection of units to be taught at each institution was determined by the teacher's choice and expertise.

The nature and extent of programming in BTEC IT

Four out of the five BTEC IT students interviewed had taken a programming unit. However, there were variations in programming languages and style of unit delivery. One student took a Java programming unit during his first year but could not remember which concepts were taught or how it was assessed. Two other students programmed in the C# language during their BTEC IT course. One student said that his programming tutor taught the constructs of sequence, selection and iteration, but did not cover functions or data structures. Another student took both Procedural Programming and Object-Oriented Programming (OOP) units. OOP extended the syllabus of Procedural Programming by teaching concepts of inheritance, encapsulation and polymorphism. Another student took a functional programming unit which concentrated on creating MS DOS programs in the C language. He added that this unit only covered basic concepts such as variables, loops and selection statements. This also corroborated the testimonies of other students.

A survey of previous programming experience (refer to Appendix 13 for the survey questions) was completed by 61 students attending L4 studies in the 2016/17 academic year. Out of the 61 students, 36 students had some previous exposure to programming and 14 had taken a BTEC IT course. Table 4.13 below illustrates that most BTEC IT students understood how to assign values to variables (100%), produce console output (100%) and use selection statements (93%). A smaller percentage of students were comfortable with array constructs (36%) and writing functions (36%) that accepted parameters (21%) and returned values (21%). The questionnaire responses therefore triangulated interview findings regarding the programming concepts typically taught in BTEC IT.

Table 4.13. Knowledge of programming topics amongst BTEC IT students before starting Programming Concepts.

Concepts	The percentage of students with prior knowledge of programming concepts
Output of data to screen	100%
Declaring and assigning values to variables	100%
Creating loops	57%
Writing if statements	93%
Writing functions	36%
Parameter passing	21%
Returning values from functions	21%
Declaring a one-dimensional array	29%
Input and output of arrays	36%
Writing classes and creating objects	43%
Inheritance, Polymorphism and Encapsulation	29%

Notes: (1) 14 of the 61 who participated in the survey were BTEC entrants.

The effectiveness of BTEC IT in preparation for L4 programming

For most students BTEC IT represented their first experience of programming. Therefore, gaining some experience in applying basic constructs (such as sequence, selection and iteration) was valuable preparation for undergraduate study. Most students agreed they would have benefited from also having experience of functional design, data structures and object-oriented principles before university. Some students were able to take an object-oriented principles unit, but this depended on the institution and the teaching expertise available.

Other notable findings regarding BTEC IT

Three BTEC students commented that their colleges allowed students to resubmit their coursework. One student said his tutors provided feedback on work submitted before the deadline, which enabled students to improve their grades. Another student commented that he was allowed one opportunity to receive initial feedback on his submissions before being marked for a finalised grade. Opportunities to improve submissions, guided by feedback, might explain why the proportion of students achieving top BTEC grades is greater than that for A-level entrants (HEFCE 2015). This assessment practice is also commonplace for other BTEC awards; Carter and Bathmaker (2017) thought this had a detrimental impact on proficiency in BTEC Engineering courses. Although A-level students may re-sit their exams, they do not receive personalised feedback, nor are they allowed to resubmit amended scripts. They can only sit a new exam with unseen questions.

Most BTEC students chose to study a BTEC qualification to avoid being assessed by exams. One student chose a BTEC qualification because he did not consider himself “academic enough” to take A-levels and believed that this was because he did not perform well in exams. Other students commented that they “hated exams”, “never did well” or felt “too much pressure” when taking them. Such attitudes towards examinations may partially explain the growing popularity of BTEC qualifications as entrance qualifications for UK universities (HEFCE 2015; Reidy 2015) and is also apparent in Figure 4.3.

Although most interviewees thought their BTEC IT tutors explained concepts well and were supportive, two students had negative experiences. One student said that his tutors did not support him or others in his class. This gave him and his peers little aspiration to achieve and contributed towards disruptive behaviour in the classroom. He added that this institution was one of the poorest achieving colleges in the county. The other student commented that his tutors “didn’t care”, and “didn’t teach us”.

BTEC Games Development

BTEC Games Development is a specialised version of the BTEC Creative Media Production course. Students can also choose to specialise in other subject areas, if offered by the institution, such as: Television and Film; Radio; Sound Recording; Print-Based Media; and Interactive Media (Pearson 2010b: vii).

The number of students who had studied BTEC Games Development was small in the sampled population (at 4) when compared to that for entrants holding the BTEC IT Qualification (at 26) (Figure 4.1). Although all student participants were invited, only one student agreed to be interviewed. This student had initially considered studying A-levels, but he was dissuaded when his brother informed him that A-level ICT concentrated on database management. This did not appeal to him and he instead decided to study the Games Development BTEC course at a local college. This also coincided with his passion for video games.

The nature and extent of programming in BTEC Games Development

The BTEC Games Development course featured a programming unit, providing an opportunity for the one student interviewed to learn to program in Visual Basic (VB) using the Microsoft™ Visual Studio™ IDE (Integrated Development Environment). He was taught basic constructs (sequence, selection, iteration) and also learned some object-orientated concepts. The assessments for this programming unit required the design of data entry forms and business applications.

The BTEC Games Development syllabus (see Appendix 14) specifies seven mandatory units and 15 optional units (Pearson 2010c: 39, 49); considerably fewer options than the 40 units for the BTEC IT course. The syllabus specifies a mandatory Computer Game Platforms and Technologies unit, as well as optional units in Object-Oriented Design for Computer Games and Computer Game Engines.

The effectiveness of BTEC Games Development in preparation for L4 programming

The interviewed student thought the BTEC course gave him a good understanding of programming concepts. However, he acknowledged that he had gaps in his knowledge before starting the undergraduate introductory programming course. He disagreed with the suggestion that such gaps may be due to inadequate teaching at BTEC level. He thought he only required some additional time to more fully understand concepts, and that the knowledge gaps had for the most part been addressed after revisiting concepts on the undergraduate introductory programming course. He thought he would have struggled in the first year had he not been exposed to programming during his BTEC Games Development course.

A-level ICT

A-level ICT has been offered in various forms in the UK since the 1990s (Wells 2012: 9). Currently, there are large differences between the curriculums offered by different exam boards. Edexcel's ICT qualification is available as a double award or single award qualification. The double award allows students to take 12 units from 14, whereas the OCR Board's single award comprises four units; two units assessed at AS level, and two at A2 level.

The sample of students who studied A-level ICT was relatively small compared with that for BTEC IT students. However, the students who participated in interviews clearly had widely ranging experiences of this qualification.

The nature and extent of programming in A-level ICT

One student said that no programming or algorithmic development was taught during his A-level ICT course. He described the ICT qualification as "irrelevant" to the computer science course he went on to study at university.

Another student who took A-level ICT commented that his curriculum did not include a dedicated programming unit. He did a little coding in Visual Basic (VB) which involved data input and output. However, this mainly involved dragging and dropping form controls and coding them to respond to click events. His tutor only briefly mentioned variables and sequence and the other basic constructs such as selection and iteration were not taught. He went on to say that the course was more concerned with "why we make and use software than programming the software." Units which include words such as 'software' and 'applications' in their title (OCR 2013: 6) often explore reasons for using MS Office™ applications and other software applications in place of teaching programming skills (OCR 2013: 9, 11). The full list of units featuring in OCR's A-level ICT curriculum are recorded in Appendix 15.

Another student had a more positive experience of studying the A-level ICT. He took Edexcel's double award ICT qualification, which featured an optional programming unit (also listed in Appendix 15). In this unit he programmed in VB and C# in the Visual Studio™ environment. He also had additional programming experience of developing applications for personal and professional use. As a result, he already possessed a strong grasp of the foundational principles of programming. This enabled him to succeed in the first year of undergraduate study.

The effectiveness of A-Level ICT in preparation for L4 programming

The student who studied the double award of A-level ICT described his programming tutor as being "excellent." He commented that "my tutor had many years of programming experience from working in industry and was able to pass on his expertise." He went on to say that "other students in the class – who did not have experience - also said that he was excellent at helping them to learn programming for the first time." Whilst the student was positive about his programming tutor, he did not say the same for some of his other tutors. This led him to think that the effectiveness of a qualification in preparing students for the next level of education "depended upon the teacher" more than the syllabus or qualification.

Another student was not sure that A-level ICT gave him "much of an advantage" over students who had not taken the course in preparation for undergraduate level computing courses. He saw ICT as distinct from computing, and therefore, an unrelated subject. He compared the difference between ICT and computing to algebra and statistics in mathematics. However, he found that being assessed through coursework at A-level helped him to present coursework at undergraduate level.

A-level Computing

The sample of students who studied A-level Computing was also small compared with the population of BTEC IT and A-level ICT students. However, there is more consistency between the curriculums offered by exam boards for A-level Computing than for A-level ICT. Most exam boards specify four units for their syllabus; two units assessed at AS level and two units at A2 level (Appendix 16).

The nature and extent of programming in A-level Computing

Programming did not always feature in A-level ICT and BTEC IT but it is strongly emphasised in A-level Computing. The syllabuses for OCR and AQA A-level both include a mandatory unit on programming at AS and A2 level (AQA 2014a: 5; OCR 2015: 6). Most BTEC IT students covered only the basic constructs (sequence, selection and iteration), but A-level Computing syllabuses specify that these constructs should be taught in the AS level accompanied by functional design and arrays (AQA 2014a: 6). The A2 level unit then extends this with topics such as OOP, additional data structures and searching and sorting algorithms (AQA 2014a: 11 - 12). Both AS and A2 programming units are accompanied by the teaching of problem-solving strategies (AQA 2014a: 5, 11). OCR's (2015: 13, 24) AS and A2 units also include problem-solving strategies and software development lifecycles. At the time of writing, Edexcel does not currently provide A-level Computing, despite offering A-level ICT and GCSE Computing.

The effectiveness of A-Level Computing in preparation for L4 programming

Unfortunately, most A-level Computing students declined to participate in an interview. One student who took A-level Computing commented that his course was disrupted as "my teacher left a few months after I started." He went on to say that "we ended up having to teach ourselves, which wasn't great." However, the course content covered programming in great depth: "we spent most of our time practising to program and develop applications which taught us a lot about problem-solving." Therefore, by devoting half of the curriculum to teaching programming concepts and problem-solving techniques (compared to one unit in BTEC courses and A-level ICT), A-level students commence their L4 studies with a deeper understanding of programming principles.

Other notable points regarding A-level Computing

Some A-level ICT students wanted to study A-level Computing but most found that the subject was not offered at their school. Students were commonly told that the institution did not have a teacher with sufficient expertise. Several students thought this was because industry offered higher salaries than teaching positions, hence being a disincentive for computer science graduates or employees to train and qualify as teachers. One student commented that ICT teachers were expected to teach other subjects at his school and there were no teachers assigned to teaching ICT exclusively.

This is consistent with national reports on the shortage of educators able to teach A-level Computing (Johnson 2013; Marshall 2013). However, some schools did not offer the subject due to poor uptake. As indicated in Section 1.2, previous GCSE level ICT curriculums had created a negative impression of computing subjects in general and often discouraged students from studying the subject further. Despite this, the national statistics regarding completion rates indicate recent increases in the numbers of students enrolling on A-level Computing courses (Table 4.14).

Table 4.14. Completed A2 A-level Computing and ICT qualifications between 2013 and 2016.

	2013	2014	2015	2016
A-level Computing	3758	4171	5383	6242
A-level ICT	10419	9479	9124	8737

Notes: (1) Data taken from GCE statistics references: (Joint Council for Qualifications, 2013, 2014, 2015, 2016a).

Summary of Section 4.1

This section summaries the key findings of the mixed-methods investigation into the L4 programming performance of BTEC and A-level students.

A-level students achieved a significantly higher average L4 programming grade than BTEC students (Figure 4.2). A-level students also achieved a slightly higher average L6 programming grade than BTEC students (Table 4.5). However, the UCAS points of BTEC students aligned more strongly with L4 and L6 programming grades, than for A-level students (Table 4.1 & Table 4.4). A-level ICT students achieved a similar median L4 programming grade to A-level Computing students, but the range of scores was far greater (Figure 4.1). More BTEC students applied to study computing related undergraduate courses than A-level students, corroborating national trends (HEFCE 2015; Reidy 2015).

The equivalent total BTEC points of Sri Lankan students correlated moderately strongly with L6 Programming grade (Table 4.3). However, the individual BTEC Computer Programming unit was only weakly related to L6 Programming grade (Table 4.9). Both UK and Sri Lankan BTEC courses tend to offer one (occasionally two, depending on institution) programming unit. However, this only constitutes one out of approximately 16 units covered over two years.

Interviews and questionnaire data revealed that UK BTEC IT courses tend to cover only the basic programming concepts (sequence, selection and iteration). Similarly, most A-level ICT courses rarely cover programming concepts; only one student taking a double-award ICT qualification by Edexcel, was taught programming and this was also because he had a tutor with programming expertise.

On the other hand, A-level Computing syllabuses devote over half of their curriculum to teaching programming and problem-solving concepts. However, A-level Computing is not offered in many institutions due a lack of teachers with sufficient expertise. Therefore, it appears that the extent of programming coverage does depend heavily on institutions employing teachers with appropriate expertise.

4.2 Further analysis of applicant data

4.2.1 Analysis of trends between applicant data and programming grade performance

Methods

The quantitative analysis in this section investigates the relationships between L4 grade performance and applicant profile data such as gender, DOB, Local Education Authority (LEA) catchment, nationality, ethnicity, and degree course. As these attributes are known in advance of undergraduate entrance, discovery of relationships would enable early intervention. The use of notched boxplot analysis is continued in this study to visualise and determine statistical significance between different categories of data. Pearson's correlation analysis is also applied to continuous variables such as age and grade to determine their relationship.

This data was obtained for the identical sample of academic years 2013 – 16 analysed in Section 4.1. All non-numerical data was quantised for correlational analysis. Student age, at time of commencing study, was calculated from DOB. Other attributes such as gender, degree course, nationality, ethnic background and LEA data were assigned numbered categories (non-weighted). Students applied from a wide range of geographical regions in the UK. London boroughs were, however, grouped together for comparison with Buckinghamshire and surrounding regions.

As previously stated in Section 4.1, international differences meant that some comparisons could not be performed due to insufficient or incompatible data. The LEA catchment data is a UK metric and does not exist in Sri Lanka. Term addresses were recorded but not locations of previous education. In addition, every student enrolled at SAITM was Sri Lankan, thus a comparison of performance differences between nationality groups was not possible with the Sri Lankan dataset. All SAITM students were enrolled on the same degree course - BSc Hons Computing (Interactive Media) – also nullifying a comparison of performance differences.

The programming performance of men and women

Notched boxplot analysis (Figure 4.4) demonstrates that the median L4 programming grade was greater for L4 Bucks men (62%) than for women (56%). This sample contains a disproportionately high number of men to women, which is reflective of findings reported in Section 2.1 (Carter & Jenkins 2001; Universities UK 2015: 23). Whilst some women are prone to withdrawing from computing related courses (Lewis et al. 2006: 6; Margolis & Fisher 2002: 80), Figure 4.4 demonstrates that all women attempted L4 programming assessments. In contrast, some men did not submit assignments (they were given '0' for a non-submission).

L6 Sri Lankan women (70%) achieved a higher average programming grade than their L6 male peers (62%) (Figure 4.4). Notched boxplot analysis illustrates that L6 Sri Lankan women (70%) had a significantly higher average Programming Grade than L6 Bucks women (43%) (Chambers et al. 1983: 62). L6 Sri Lankan women (70%) also had a higher median grade than L4 women (56%) (although not regarded to be significant). However, median grade performance amongst men is more consistent.

Figure 4.4 also illustrates a wider range of grades for L4 men and women compared to L6 men and women. Grade distribution of Bucks L4 programming courses (Bucks 2015a; Bucks 2015b; Bucks 2016a; Bucks 2016b) is consistent with bimodal distributions commonly reported in literature (Robins 2010: 4). Whereas L6 grade distributions tend to be lower, as low achieving L4 and L5 students generally do not progress to L6.

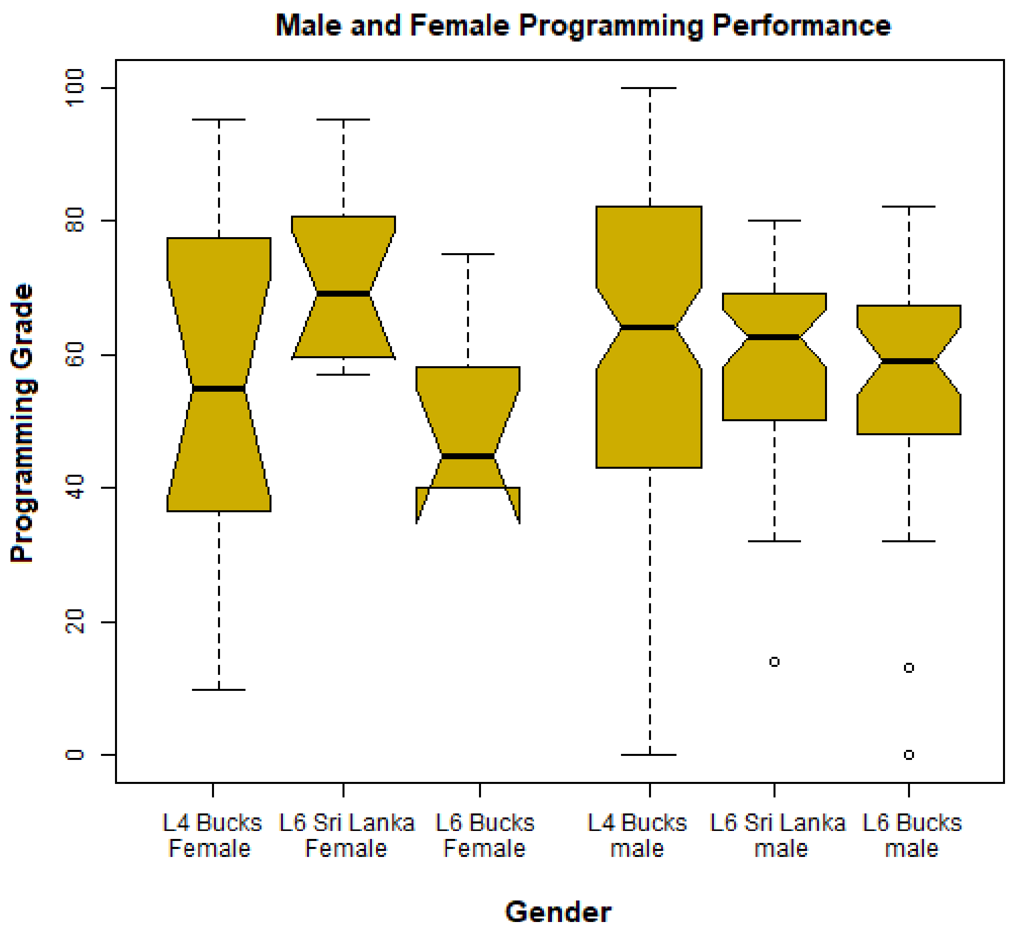


Figure 4.4. Notched boxplot comparisons of programming grades of L4 & L6 Bucks and L6 Sri Lankan students. Notes: (1) n for L6 Sri Lankan categories is: Female, 12, Male, 50; n for L4 Bucks categories is: Female, 15, Male, 105; and n for L6 Bucks categories is Female, 8, Male, 35.

The usefulness of student age as a predictor of programming success

Figure 4.5 illustrates the age distribution of Bucks L4 students. Most students are aged between 18 and 21 (102 out of 120; 85%), and these vastly outnumber mature students (older than 21) (18 out of 120; 15%). This ratio is consistent with reports of rising applications from young students, and declining mature student applications (Department for Education 2016; Independent Commission on Fees 2015: 17).

Pearson’s Correlation analysis revealed a slightly negative but significant correlation (r = -0.202, p = 0.027, n = 120) between student age and programming performance. This suggests that younger students achieve marginally higher grades than mature students.

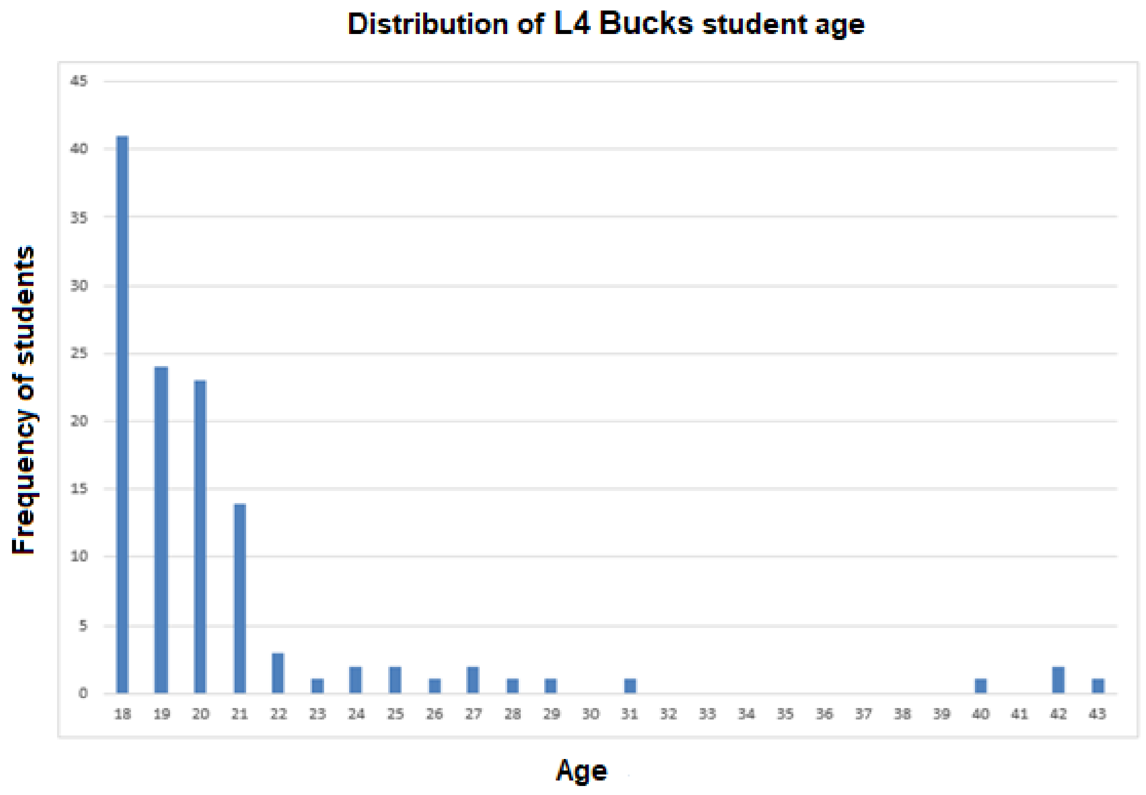


Figure 4.5. Age distribution of L4 Bucks students.
Notes: (1) n = 120.

Figure 4.6 illustrates the age distribution of L6 Sri Lankan students. In addition to entering two years later than L4 students, Sri Lankan students are required to complete a year of bridging work in preparation for undertaking L6 courses. Consistent with the distribution of L4 Bucks entrants, the number of younger L6 Sri Lankan learners vastly outnumbers the population of mature learners.

Interestingly, a few students are aged between 18 and 19, which is younger than the expected age of L6 entrance in the Sri Lankan cohort. However, there are exceptional circumstances in which some Sri Lankan students start compulsory education one or two years earlier than their peers (No author, n.d.), which may explain this pattern.

Pearson’s Correlation analysis reveals a positive but weak and non-significant relationship (r = 0.200, p = 0.119, n = 62) between student age and L6 programming grade. This suggests that some mature students perform marginally better than younger students. Despite the weak correlation, this positive relationship contrasts the negative relationship with Bucks data (r = -0.202, p = 0.027, n = 120).

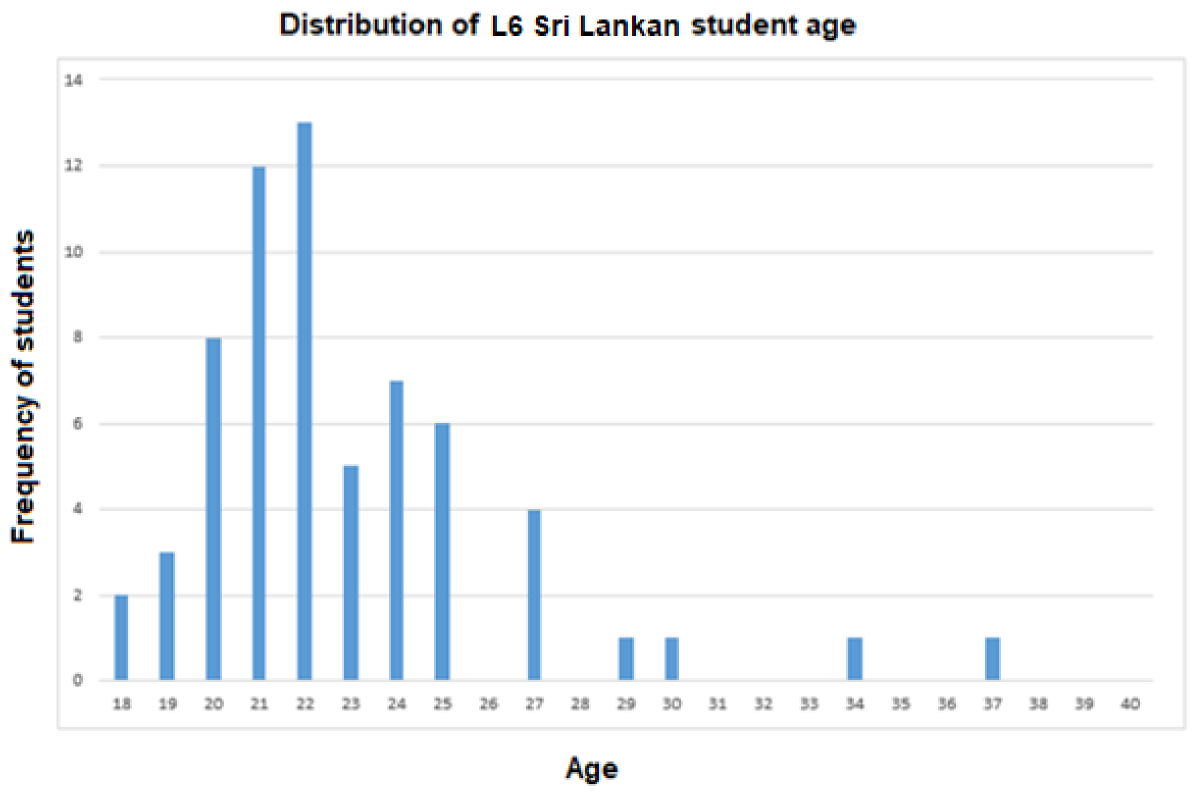


Figure 4.6. Age distribution of L6 Sri Lankan students.
Notes: (1) n = 64.

LEA, Nationality, Ethnic Background

Figure 4.7 illustrates the distribution of programming grades across LEA (Local Education Authority) boundaries - the catchment area where students completed their previous study. Some geographical regions represented by one student were omitted from analysis. Whilst larger sample sizes are desirable, regions featuring a small number of students were included for comparison with more populous areas. The largest population of students applied from the county of Buckinghamshire (in which Bucks New University is located). Train and bus links also enable London students to commute, which might explain why it is the second largest sample. However, notched boxplot analysis illustrates that the average grade for London students (48%) is lower than that for Buckinghamshire students (73%) (Figure 4.7).

The median grade was significantly lower for Slough (40%) and Milton Keynes (26%) students than for most regions. Further investigation revealed that students from these two regions were previously taught by the researcher. Factors such as low confidence, poor time management, and lack of integration with peers are thought to influence performance more than LEA.

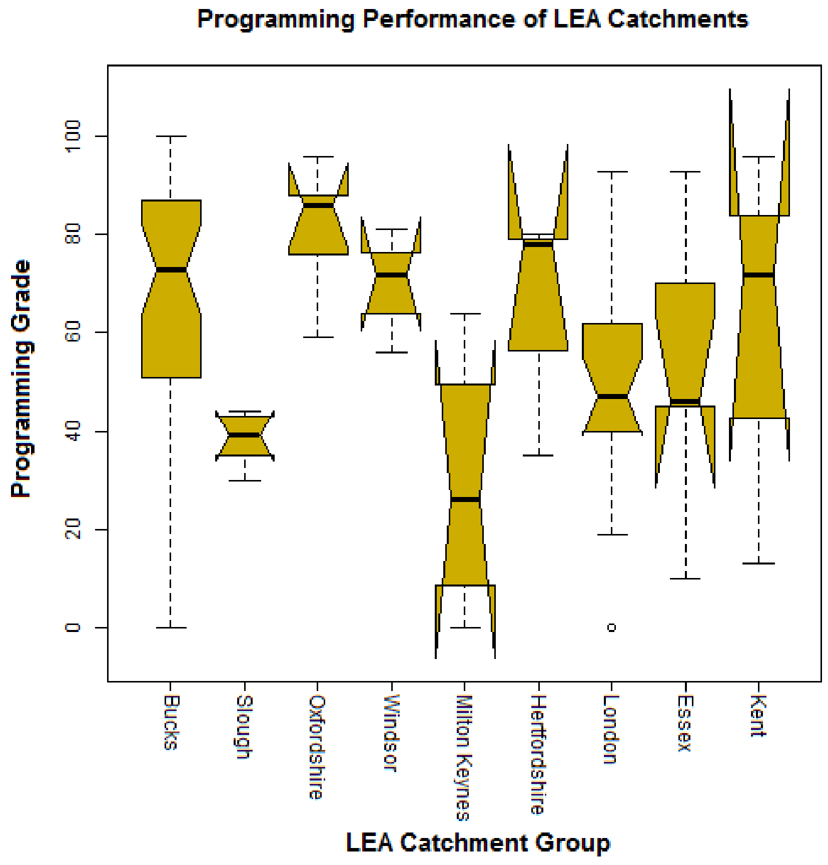


Figure 4.7. Notched boxplot comparisons of programming grade across locations of previous study. Notes: (1) n for categories is: Bucks, 38; Slough, 6; Oxfordshire, 5; Windsor, 3; Milton Keynes, 4; Hertfordshire, 3; London, 19; Essex, 5 & Kent, 3.

Most students in this sample were born in the UK, and Figure 4.8 illustrates minimal difference between the median programming grade of UK (62%), European (70%) and Asian (66%) students. However, African students (median of 30%) scored significantly lower grades (Chambers et al. 1983: 62) than the other three nationalities. Despite this comparison being significantly different, the sample size of five African students is too small to conclude that all African students are vulnerable in programming courses. However, it is notable for further investigation that three of these five students are mature students and it is known that such students have difficulty integrating with traditional HE environments; consistent with literature (Gordon 2016: 6).

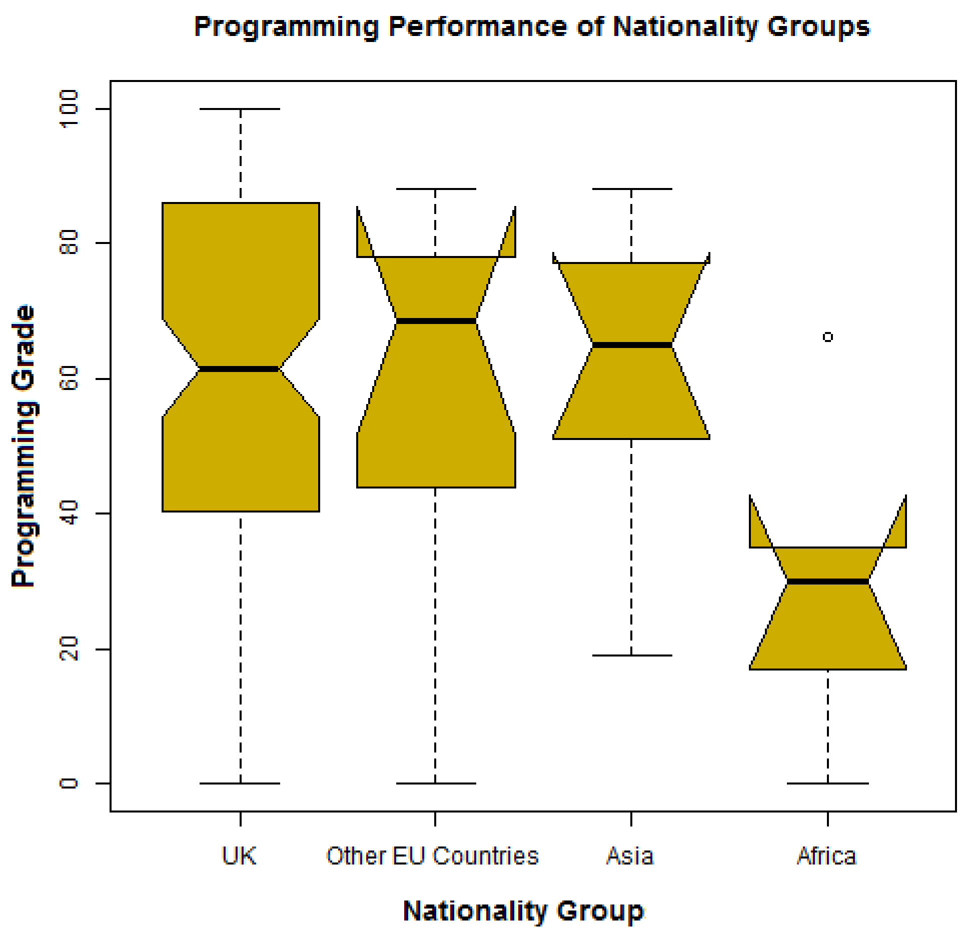


Figure 4.8. Notched boxplot comparisons of programming grades and nationality categories.
Notes: (1) n for categories is: UK, 96; Other European countries, 10; Asia, 9 & Africa, 5.

Although most students are white, there are sizeable populations of 'black and ethnic minority' (BME) students. This is consistent with national statistics which report that undergraduate computer science courses often attract higher than average numbers of BME students (CPHC 2012; Gordon 2016: 6). Notched boxplot analysis in Figure 4.9 indicates that white students (72%) achieved a higher median programming grade than Asian students (56%). This also corroborates national attainment reports (CPHC 2012: 9; HEFCE 2014: 3). White students, however, did achieve a broader range of programming grades compared with Asian students (Figure 4.9).

The modest number of Chinese students (77%) and students of 'Other Mixed Background' (87%) achieved the highest median grades of all ethnic categories. This would appear to support western stereotypes regarding Chinese attitudes towards work (Phillips 2015) although this does not account for individual variance.

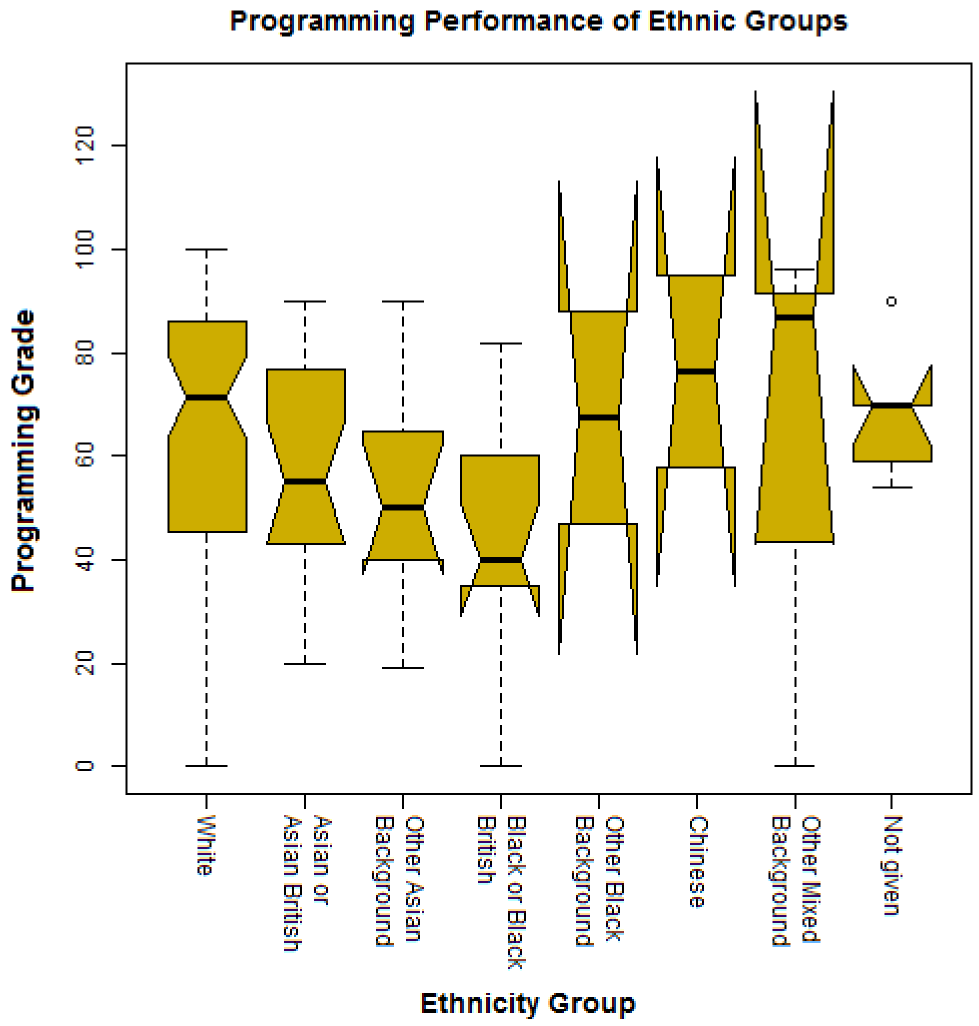


Figure 4.9. Notched boxplot comparisons for programming grade across ethnicity categories.
Notes: (1) n for categories is: White, 64; Asian or Asian British, 21; Other Asian Background, 10; Black or Black British, 13; Other Black Background, 2; Chinese, 2; Other mixed background, 3 & Not given, 5.

Comparison of Degree course (choice of)

Each of Bucks' six degree courses emphasise a particular aspect of computing but all students are required to complete the introductory programming course in their first semester. Independent Games Production was introduced in the 2015/16 academic year, which may explain the smaller number of students for their sample. However, the Software Engineering degree course was being delivered when the 2013/14 and 2014/15 samples were collected.

Notched boxplot analysis in Figure 4.10 illustrates consistent grade ranges and median scores across most degree course programmes. None of the Independent Games Production group failed the course. However, this finding may reflect one cohort of students (2015/16) rather than the course itself. Figure 4.10 illustrates that the median grade for the Software Engineering students (42%) is considerably lower than the other courses too. However, further investigation revealed that three students (of nine in the sample) had already been identified to have performed poorly for reasons unrelated to LEA, and may therefore also distort the typical performance of Software Engineers.

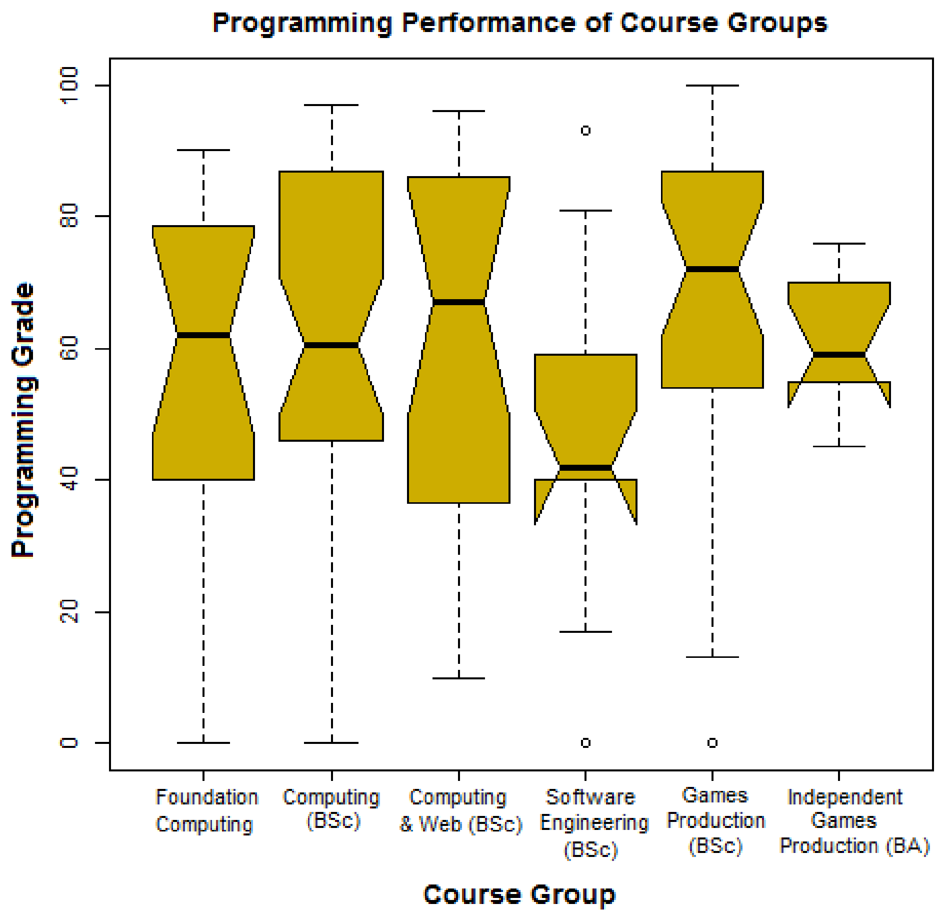


Figure 4.10. Notched boxplot comparisons between programming grade and course categories.
Notes: (1) n for categories is: Foundation Computing, 16; Computing (BSc), 38; Computing & Web (BSc), 20; Software Engineering (BSc), 12; Games Production (BSc), 25 & Independent Games Production (BA), 9.

Summary of Section 4.2

This section discovered that the range of L4 programming grades for men and women is significantly greater than that for L6 men and women (Figure 4.4). This greater range reflects the bimodal distribution reported in literature (Robins 2010); the population of those who fail is as great as those who achieve over 70%. Fewer students fail at L6 due to the requirement to pass L4 and L5, and the optional L6 programming courses tend to attract the most motivated and able students. Figure 4.4 illustrates that the average grade for L6 Sri Lankan men is consistent with the grade for L6 Bucks men; however, L6 Sri Lankan women achieved a significantly higher grade than L6 Bucks women. However, this is likely to be due to individual circumstances that caused L6 Bucks women to underperform, and exceptional L6 Sri Lankan women who outperformed their male counterparts.

There are significantly greater populations of young students than mature students in both the L4 Bucks dataset and the L6 Sri Lankan cohorts (Figure 4.5 & Figure 4.6). This reflects the difficulty of recruiting and retaining mature students (Gordon 2016: 6; Independent Commission on Fees 2015: 17). Although Pearson's correlation analysis demonstrates a weak correlation between age and grade for both cohorts, the correlation was positive for L4 cohorts and negative for L6 cohorts. A possible explanation is that more mature students retook L4 courses than for L6 courses. However, these correlations are too weak to consider age as an indicator of programming grade.

Comparison of LEA attributes identified that Slough students achieved significantly lower grades than other regions (Figure 4.7). However, further investigation discovered that course engagement factors offer more explanatory value for poor attainment than the region of prior study. Notched boxplot analysis also identified that African students achieved significantly lower grades than their UK, Asian and European peers (Figure 4.8). However, this sample of African students comprised mostly mature learners previously identified to perform poorly for various reasons. Figure 4.9 illustrates that BME students are often outperformed by white students, which corroborates national reports (Gordon 2016).

Finally, notched boxplot analysis (Figure 4.10) demonstrates consistent performance across most degree programmes, except for Software Engineering. But as previously established, the Software Engineering sample features mature students and other young students identified for their poor performance. It is notable that Games Development students achieved the highest median grade. Section 5.3 goes on to explore this trend further and attributes their performance to behaviours representative of social learning theories (Lave & Wenger 1991; Wenger 1998); consistency, working discipline and belongingness (Thomas 2012a).

4.3 Chapter 4 Conclusion

This chapter set out to investigate whether L4 programming performance differences existed between A-level and BTEC entrants, in addition to evaluating the usefulness of typical applicant data for predicting L4 performance. This chapter addresses Objective 3 "E*valuate the usefulness of prior qualifications for preparing students to learn computer programming in a higher education establishment*" as stated in Section 1.5.

Section 4.1 did identify patterns of performance unique to A-levels and BTEC entrants. However, A-level students in general (both A-level students who took ICT or Computing, and A-level students who did not) achieved higher L4 programming grades than their BTEC peers. Although small sample sizes do limit the validity of findings related to A-level Computing and A-level ICT, the total number of A-level students and BTEC students sampled in this study is similar in size (40 each) (Figure 4.2). This merits further investigation to discover what factors of each qualification explain this trend.

However, this chapter also found minimal difference between the L4 programming performance of those with prior computing or ICT related qualifications and those without. It is also striking that most A-level entrants - identified to achieve a grade higher than BTEC entrants - did not study Computing or IT subjects. Qualitative analysis of interview transcripts found that prior experience does not guarantee a strong understanding of programming concepts due to varied teaching quality and content covered at each institution. Moreover, both of these findings suggest that specialist programming knowledge or experience is not required for success in L4 programming courses. Chapter 5 contributes additional evidence to triangulate this finding.

Section 4.2 discovered that patterns of performance are generally consistent with the published literature; in terms of BME and mature learners as well the wide range of L4 grade that reflects a bimodal distribution. However, whilst notched boxplot analysis identified trends of performance amongst other categories (LEA, nationality, and degree course choice), further investigation found that these samples tended to contain those already identified to perform poorly due to other reasons.

Thus, whilst some measurements of L3 performance and student profile data included on undergraduate applications may indicate L4 performance, alone, these are too reductionist (Thomas 2012a: 4). Therefore, the next chapter evaluates the indicative and explanatory value of measures of engagement such as L4 attendance, performance across all L4 courses, and peer integration and communication.

Chapter 5 Exploration of student engagement with course materials

This chapter sets out to evaluate the influence of course engagement metrics (attendance in addition to summative and formative grades) and typical student behaviours (peer communication and integration, classroom and independent study habits) on the L4 programming grade. Section 5.1 reports a multivariate analysis of course evaluation survey responses. These findings motivated further quantitative investigation regarding the impact of attendance on the L4 programming grade, and the relationship between related L4 and L6 course grades at Bucks and Sri Lanka (Section 5.2). In Section 5.3, the findings of Section 5.1 are also extended with qualitative investigation of video recordings and interview transcripts regarding the influence of collaborative behaviours and environmental factors on learning. Section 5.4 evaluates the findings of Sections 5.1 – 5.3 to determine the usefulness of course engagement metrics for predicting L4 programming grade.

5.1 Multivariate analysis of course engagement metrics

Methods

This study applies a novel technique of canonical gradient analysis, pioneered in ecological sciences, with the aim of exploring student performance and behaviours while undertaking formative and summative tasks in an L4 programming course. This adopted an ecological perspective that, similar to the positioning of organisms along natural environmental gradients, the position that students may occupy along achievement gradients is partly determined by their learning behaviours and their engagement with learning environments. Such a technique is useful for revealing complex patterns, trends, tacit communications and technology interactions associated with a particular type of learning environment.

This approach was similar to that described by Mather (2014), thereby ensuring that findings could be compared with respect to the persistence (thereby predictability) of engagement patterns (regardless of the use of an additional learning platform and modified regime). This study endeavours to determine whether the patterns of engagement and learning progress reported more recently are consistent with those reported for 2014, and thereby indicate whether earlier observations are likely to be representative of future cohorts.

Since Mather's earlier study, assessment criteria of the L4 programming courses involved were altered in an endeavour to encourage the development of the deeper understanding of programming skills required by industry. In the 2013/14 academic year students were required to complete and document all tasks (numbering 100 or so problems) issued in 'study packs'. These comprise some five to seven in-class practical exercises and one to three independent studies each week (across a 15-week semester). In the latter part of the course, these exercises are replaced with a single in-depth and extended project. During the next academic year (2014/15), students were requested to submit only independent studies and projects, significantly reducing the volume of work required for formal summative assessment. It was hoped that this would allow students greater time to concentrate on completing tasks to a higher standard. The course team also introduced voluntary formative testing measures to help students consolidate subjects covered in theory and practical sessions.

Mather's 2014 questionnaire was slightly amended (questions presented in Appendix 1). The overall structure remained unchanged as follows:

- an initial question asking students to evaluate the 'perceived difficulty' of modules;
- five test coding questions, four requiring coding concept definitions and one to identify output from a 'broken' code;
- 22 Likert scale questions (20 in 2014) concerning learning behaviours and module acceptance attitudes.

The Likert scale questions were modified for the study covering the 2014/15 academic years. This was because, in the earlier study of 2013/14 (Mather 2014), the combined test and questionnaire had been issued after students had completed the first of two sequential L4 programming courses, based on the 'Ceebot' learning environment. In 2015, the test and survey was conducted at the end of the academic year when students had almost completed both of the two progressively linked programming courses. The proposal for this study was approved by the same internal ethics panel (Appendix 3) that approved the collection and analysis of Bucks L4 data in Chapter 4. All students that participated in this study were fully informed of the purpose of the research during classes and in writing and participants consented to their responses being reported anonymously (Appendix 1). Questions were therefore amended to capture student impressions of both Ceebot (core to the Semester 1 course) and the C# language using Microsoft's Visual Studio™ platform (utilised within the Semester 2 course). A further question was included to determine student preferences for learning programming with either Ceebot or the Visual Studio™ integrated development environment (IDE).

5.1.1 Analysis and discussion

The application and interpretation of Redundancy Analysis

Questionnaire data was transferred to a spreadsheet. Programming test data was scored for correctness and 21 opinion questions captured using five Likert categories. For the purposes of canonical analysis (see Mather (2015) for details of this procedure), independent variables (corresponding to 'environmental gradients') were represented by programming skill and knowledge and module grade. The dependent variables were represented by the Likert opinion responses and a percentage record of student attendance. The analysis was conducted using the canonical analysis software Canoco 5™. This is widely used by ecologists to conduct multivariate analyses and to reveal patterns and trends in data.

Here redundancy analysis (RDA) is applied to the data set. Among the key advantages of RDA over more commonly used ordination techniques (such as Principal Components Analysis) are:

- that ordination axes are constrained in iterative steps to describe variation in the explanatory variables of interest;
- the technique does not require assumptions of unimodality (ter Braak 1987);
- the resulting correlation biplots provide an easily interpretable and graphical summary of the most important relationships in the ordination model.

The reader is again referred to Mather (2014) and the original and updated works of Canoco specialists (e.g. Corsten & Gabriel 1976; Šmilauer & Lepš 2014; ter Braak 1987; ter Braak 1992) for more detailed description of the mathematical interpretation of vectors represented by RDA biplots.

For the purposes of visual interpretation of the biplot figures, below it is sufficient to understand that vectors or 'arrows' point in the direction of maximum variation. Variables with longer arrows have greater effect on the overall model and are generally most closely correlated with the independent variables of interest (ter Braak 1987; ter Braak & Prentice 1988). Variables and axes pointing in the same direction are positively correlated, perpendicular vectors are uncorrelated and opposing ones are negatively correlated.

With respect to negative correlations, these may only be artefacts caused by 'negative' assertions in expressing questionnaire items. In the biplots below, independent variables are indicated by red arrows and dependent ones by blue arrows.

The summary statistics for analyses of both the 2014 (2013/14) and 2015 (2014/15) data are presented in Table 5.1. Eigenvalues and other expressions for the proportion of variation explained by the first two (most important) axes, as well as for overall models, are slightly greater in 2014 than in 2015. This is perhaps due to the inclusion of further variables that were later shown to be rather weakly related to explanatory variation (notably two questions to capture preferences for the two environments used for teaching). Monte Carlo permutation tests indicate that model axes significantly describe variation in dependent variables.

Table 5.1. Summary statistics for RDAS Presented in Figure 5.1 and Figure 5.2.

Statistic	*Axis 1*	*Axis 2*
2015 Analysis		
Eigenvalues	0.0770	0.0258
Explained all variation (cumulative %)	7.70	10.27
Pseudo-canonical correlation	0.5968	0.6222
Explained fitted variation (cumulative %)	66.98	89.40
Permutation Test Results (on all axes)	pseudo-*F*=1.5; *p*=0.05	
2014 Analysis		
Eigenvalues	0.0915	0.0516
Explained all variation (cumulative %)	9.15	14.31
Pseudo-canonical correlation	0.8523	0.6975
Explained fitted variation (cumulative %)	54.04	84.52
Permutation Test Results (on all axes)	pseudo-*F*=1.4; *p*=0.04	

Results for 2015 and their comparison with 2014

Despite changes made to delivery, as well as differences in student cohorts and assessment regime in 2015, the overall patterns of biplot ordination for 2015 (Figure 5.1) were similar to those in 2014 (Figure 5.2). The most notable feature of the 2015 data (similar to 2014) is the poor correlation between course assessment ("Module %" vector) and 'real' programming skills "Test code skill" and "Test code knowledge" vectors, in Figure 5.1 and Figure 5.2). In fact, the increased angle (towards the perpendicular) between module assessment and test variables suggests that the modified assessment may not have had the desired effect of cultivating a deeper understanding of principles.

With regard to other relationships, all assessment indices (red) appear to remain (as in 2014) relatively strongly correlated with the 'commitment' indicators of independent study and homework (16, 19), but less so (in 2015) with the inclination to maintain a logbook of practical work (as indicated by the shorter vector for item 12 in 2015 compared to 2014). This latter observation is of some pedagogic significance because the modified assessment regime for 2015 also omitted the need to submit in-class solutions, and required students to present a smaller portfolio of independent-study tasks.

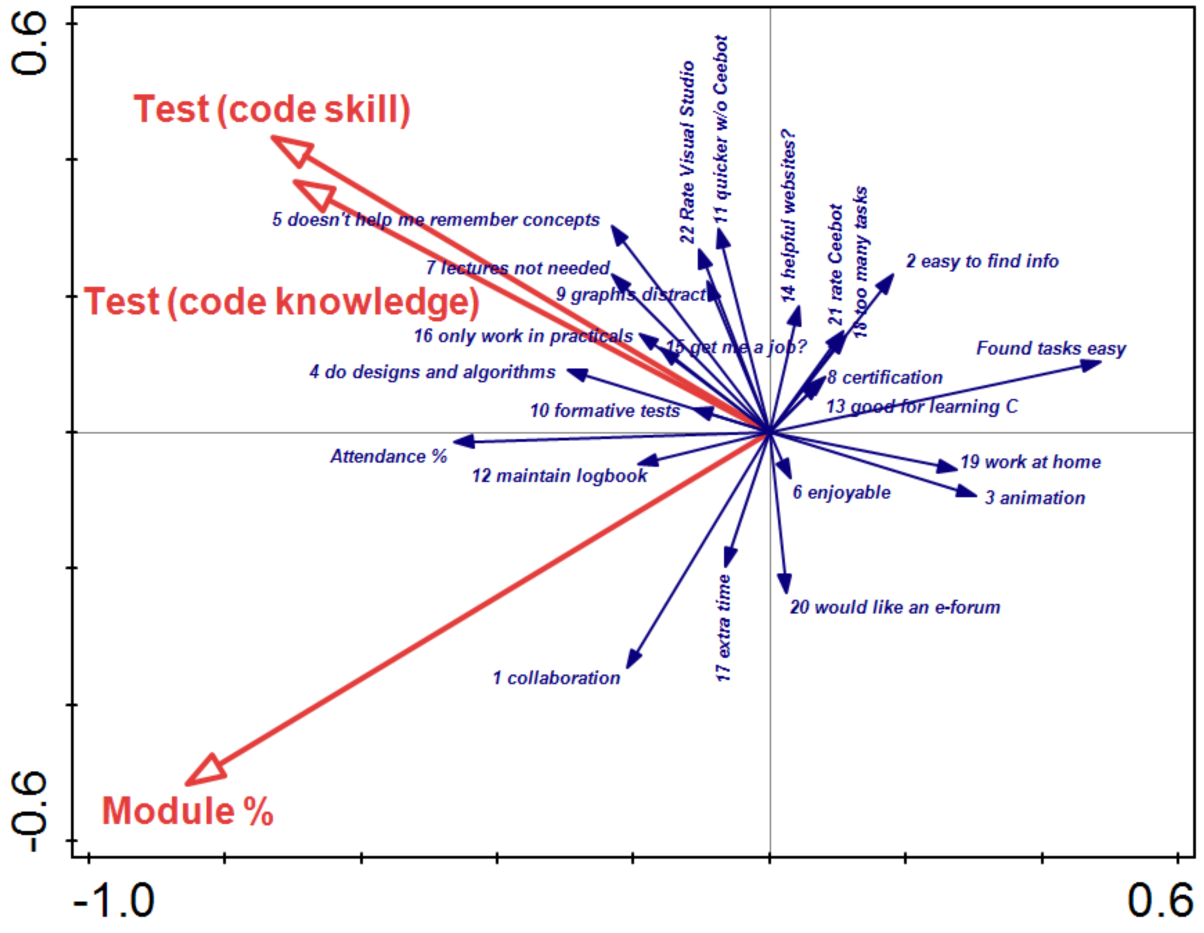


Figure 5.1. Biplot for RDA of 2015 data.
Note: response variables (student behaviour and acceptance items captured by questionnaire and attendance logs) are represented by blue arrows and with three key 'explanatory' learning performance gradients (red arrows) for Module % and tested programming knowledge and coding skill.

Other noticeable between-year pattern similarities included both the orientation and strength (length) of key variables. These included expressions: "found tasks easy" (easy=scored 1 and very difficult scored 4); item 3, Ceebot animation was helpful (1=strongly agree to 5 strongly disagree); and item 19 preference to "work at home" (1=strongly agree to 5 strongly disagree). In both years these three variables are relatively strong and closely aligned with the first axis (indicating their overall importance in the model). Their opposing orientation to assessment and test variables is only an artefact of the direction of Likert category. Thus, those students who found tasks easier (probably through greater practice) appreciated the animation used in the Ceebot learning environment and were inclined to work outside lessons and, predictably, achieved higher course and test scores. It was also unsurprising to discover that in both years, successful students had a tendency to "do designs and algorithms" (4) for their solutions; a behaviour which also directly contributes to module grades.

Of the remaining similar and influential variables, the relationships between test score vectors and the item 5 assertion that Ceebot "doesn't help me remember concepts" (and to lesser extents disagreements that it is "quicker to learn without Ceebot" and "only work in practicals", items 11 and 16), are reassuringly consistent across both years.

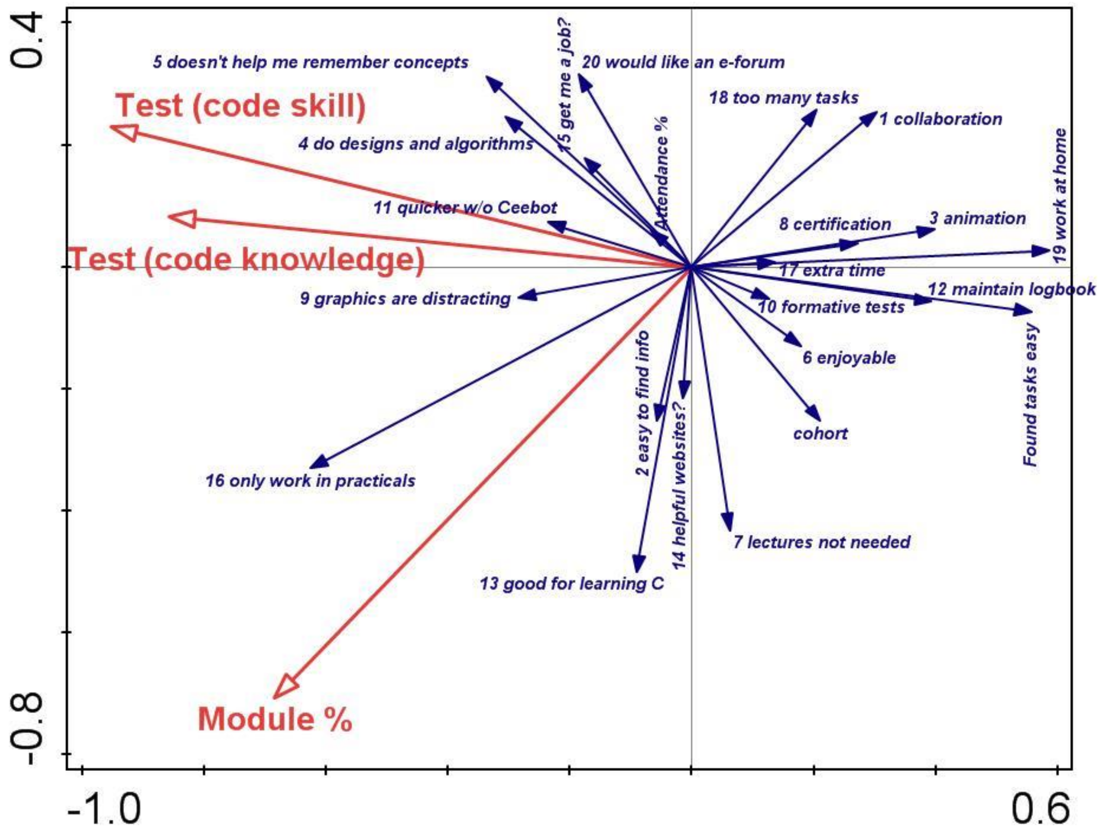


Figure 5.2. Ordination biplot from RDA of 2014 for comparison with 2015 (reproduced from Mather 2015). Note: see caption for Figure 5.1 for interpretation.

Over interpretation of weaker relationships is perhaps best avoided on the grounds that any model influences are unlikely to be significant. These include expressions for enjoyableness (item 6), the desirability of certification (8), a need to have extra time (17), the desirability of including formative tests (10), relevance for employment (15), and ease in discovering help (items 2 and 14). Similarly, analysis of questionnaire items introduced in 2015 suggests a slight preference for using Microsoft Visual Studio™ over Ceebot. It may also be inferred from the direction of relationships, that Visual Studio™ is associated with higher code skill scores, and preference for Ceebot is more strongly associated with assessment success. However, although the patterns may be consistent with such views, neither relationship is confidently demonstrated by the analysis.

Regarding meaningful differences between years, ‘attendance’ is clearly more important in 2015 and is also more closely aligned with the three learning indicators, than in 2014. This may be attributed to a small but significant cohort of unusually capable game development students who, in 2014, were able to complete programming work on a self-directed basis while only attending classes infrequently.

A changed ‘polarity’ for student collaboration

Although clearly visible in both models, the changed polarity for collaboration (item 1) is less easy to explain. In both 2014 and 2015, collaboration (the full questionnaire expression for this item was in fact “While working on exercises it is very helpful to discuss problems with friends”) is closely aligned with overall assessment success (Module %). In 2014 the opposition of collaboration and assessment vectors are consistent with a rationale that a high desire to discuss problems with colleagues (agreement with item 1 therefore low score) is associated with module success. In 2015, this relationship is such that low desire to collaborate is associated with success. It may be suggested that the 2015 group was less ‘cohesive’ than in 2014 (attendance and overall module success indicators are in fact consistent with this view), but such a complete reversal in direction of relationship indicates a need to more fully investigate the role of collaboration in learning programming.

One interesting (and pedagogically important) observation is that the complete removal of classwork from the assessed elements in 2015 has not had the desired impact of encouraging students to more deeply explore a smaller number of solutions. Instead, this appears to have substantially reduced the need for collaborative and discursive interaction towards discovering programming solutions.

Summary of Section 5.1

This extension to the work of Mather (2015) demonstrates the overall usefulness of canonical RDA as a means for exploring student progress in the educational contexts addressed by this study. Findings were also found to be useful for revealing patterns of engagement with learning materials and the class environment, as well as associating these with measures of learning achievement.

In 2015, as was the case in 2014, certain ‘commitment’ behaviours (such as the willingness to undertake homework and other independent study) are consistently associated with ‘success’. The consistent relationship of these and other ‘behaviours’ across both years suggests their reliability as ‘indicators’ of success as well as of acceptance and engagement with learning environments.

The continuing orthogonality (poor correlation) of the relationship between course grade and a student’s coding skills (despite changes made to assessments) is a matter of ongoing concern to the teaching team. Additionally, the teaching team will wish to further investigate the possibility that the reduced in-class component of assessment (intended to allow greater time to develop deeper subject understanding) may have inadvertently encouraged ‘surface’ strategies to complete independent studies at the expense of equally important (but now unassessed) collaborative classwork. It may also be concluded that analyses did not demonstrate any improvement in the alignment of assignments to workplace skills as a result of modifications to the assessment regime.

Many aspects of the observations reported here are consistent with findings reported in literature by the wider research community. Although no causality is demonstrated by the analyses here, variation in course performance nevertheless reflects the widely ranging circumstances of the student body (such as prior knowledge, ‘comfort’ in the academic environment, expectations and motivation) that are known to influence success (e.g. Bennedsen & Caspersen 2008; Richardson et al. 2012; Thomas 2012a; Wilson & Shrock 2001).

Overall many of these findings signal the importance of fully engaging with studies at all levels (attendance, collaboration, self-directed study, participation with formative class work as well as summative independent study). This is consistent with the widely accepted view that deep-level processes are required to embed learning (Marton & Saljo 1976) and that computer programming is no exception to this principle (Simon et al. 2006).

5.2 Further analysis of attendance and grade of related courses

Methods

This section continues the investigation of the sample of Bucks and Sri Lankan academic years 2013-16 analysed in Chapter 4. The same ethical standards of information provision and consent described for Chapter 4 were also applied here. Whereby Chapter 4 was concerned with evaluating the data typically present on university applications (pertaining to previous qualifications and student profiles), Chapter 5 is evaluating the data generated at university during L4 and L6, such as attendance, course grades and retake count. This study aims to evaluate the usefulness of these metrics for identifying potentially vulnerable individuals at an early stage in their respective undergraduate courses.

The use of notched boxplot analysis is continued for representing the performance or attendance of the defined categories related to L4 or L6 students. Pearson's Correlation analysis is also applied to evaluating the degree to which attendance and retake count can be utilised for predicting course grade.

Regrettably, Bucks administrators did not have access to the Sri Lankan attendance records and SAITM tutors were unable to provide this data. The comparison between the number of courses retaken and programming performance was also not viable in the Sri Lankan dataset because only three students retook a course. In these three cases, the resubmitted mark was only marginally higher than the original (e.g. 42 to 36). The original mark is used for consistency in the analysis.

In addition to evaluating the relationships between L4 and L6 grades, the use of both examinations and coursework in the L6 courses provided an opportunity to compare the average grades of these different assessment modes. The final grade for the Sri Lankan version of the L6 Advanced Interactive Programming (AIP) course is calculated from four assessments; three pieces of coursework and one exam. Sri Lanka and Bucks share the same teaching content for this course; however, Bucks assesses the course with four pieces of coursework. The Bucks' L4 introductory programming course is also solely assessed by coursework (three pieces of coursework). An Individual t-test was applied to calculate the mean and the standard deviation (spread of data) surrounding the mean.

5.2.1 The association between attendance and grade

Figure 5.3 illustrates the declining pattern of attendance for the introductory L4 programming course. This pattern of steadily declining attendance is reported to be typical for other undergraduate courses (Dolnicar et al. 2009; McKee 2014). Despite this, Pearson's Correlation analysis states there is a moderately strong and highly significant relationship ($r = 0.473$, $p < 0.001$, $n = 120$) between attendance and programming grade. For the 2013/14 academic year, students were required to submit their class exercises (exercises students were asked to work on during practical sessions). Therefore, those students who attended and participated in the practical sessions in 2013/14 attempted and completed exercises that formed part of their assignment – which may partly explain the correlation between attendance and grade. However, this relationship between attendance and grade does not account for those who worked independently of taught sessions (Mather 2014) or for 2014/15 and 2015/16 cohorts, for whom, their class exercises were not assessed.

Chapter 6 further explores methods to distinguish between those students who are capable of working independently and those who are disengaged with teaching sessions and vulnerable to falling behind. Chapter 6 also investigates the impact of course modifications on attendance.

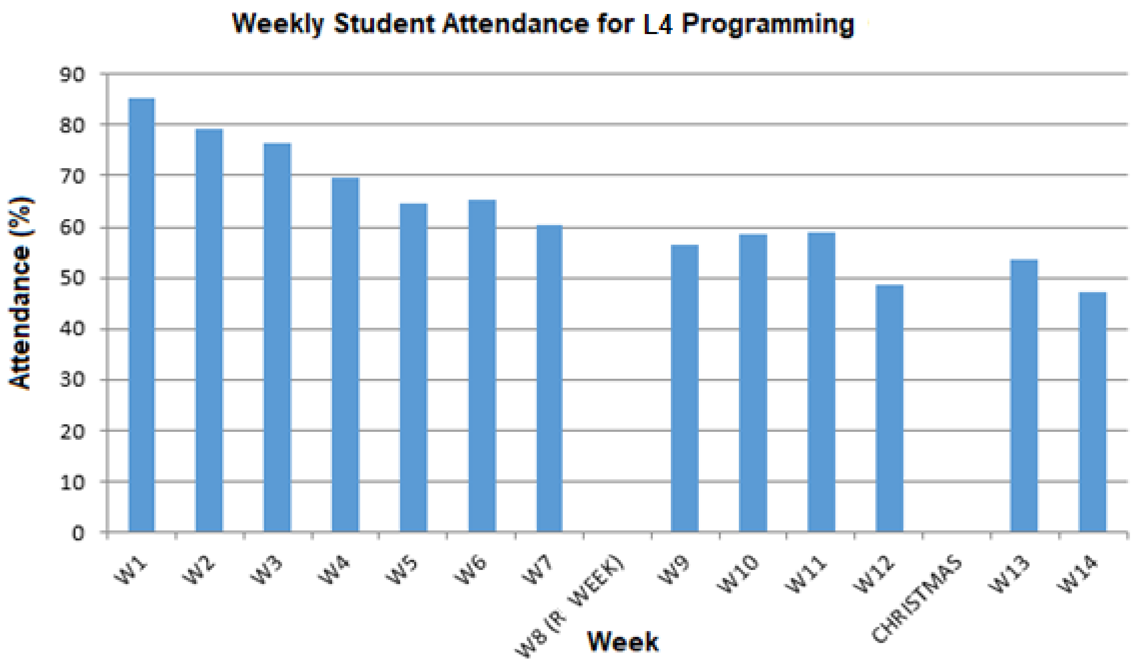


Figure 5.3. Weekly attendance for the introductory programming course.
Note: aggregated attendance percentages for students sampled from the 2013/14, 2014/15, and 2015/16 academic years.

Figure 5.4 illustrates that A-level ICT and A-level Computing entrants attended most teaching sessions (>90%) except for one A-level ICT student, whereas BTEC IT and BTEC Games students displayed a greater range of varied attendance. Nonetheless, both groups did maintain a median attendance value higher than 80%. Whilst both groups of A-level and BTEC students that did not study related computing or IT subjects had a far greater range of attendance, both groups did achieve a high median value. Therefore, previous qualifications do not appear to be influential on L4 attendance.

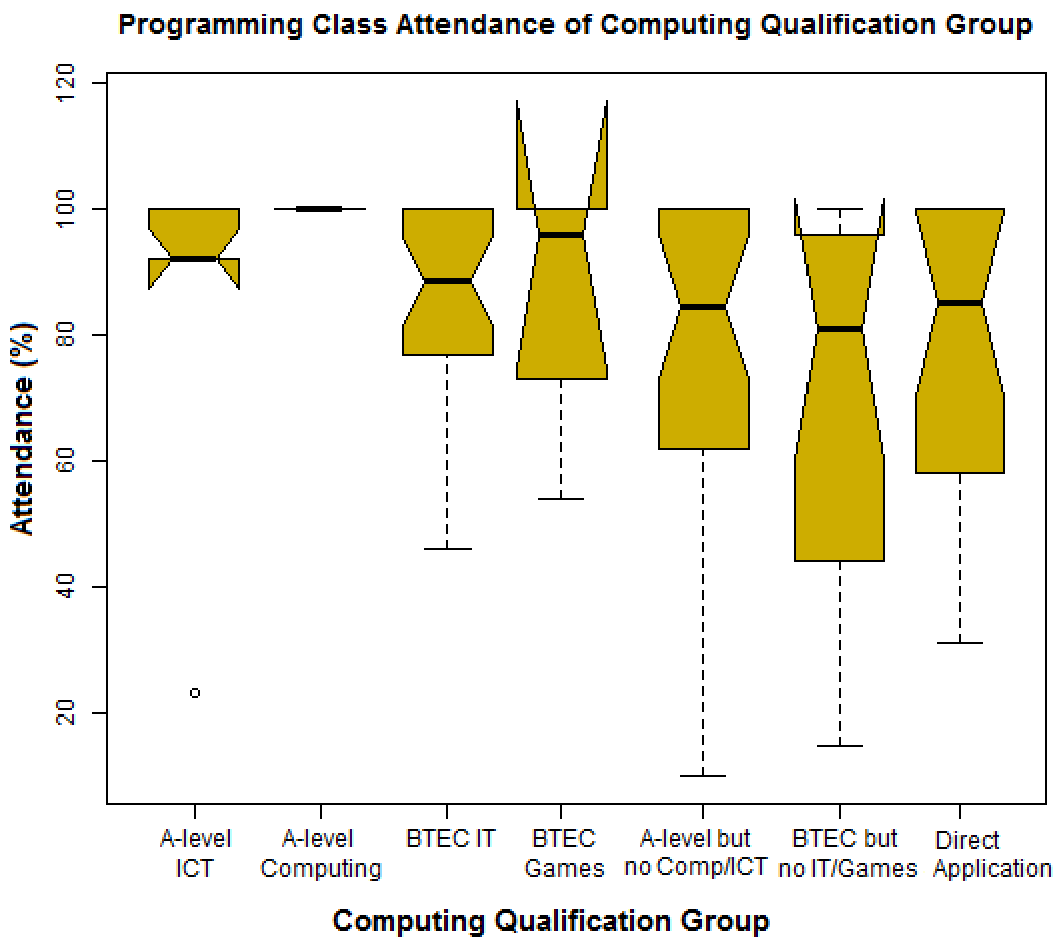


Figure 5.4. Notched boxplot comparisons between attendance and qualification categories.
Note: n for categories is: A-level ICT, 7, A-level Computing, 3; BTEC IT, 26; BTEC Games, 4; A-levels without Computing/IT, 30; BTECs without IT/Games, 16 & Direct Application, 18.

Table 5.2 demonstrates that additional comparisons between the attendance and grade of L4 courses are predominantly strong and positive correlations (with Pearson's coefficient, r, ranging between 0.366 and 0.753). The weak and non-significant relationship (r = 0.058) between the attendance and grade of the Digital Technologies course is the exception to this trend. This, therefore, merits further investigation. However, the otherwise strong relationships suggest that attendance of L4 courses is beneficial for successful grade performance.

Table 5.2. Pearson's Correlation analysis between attendance and L4 grades.

Comp Architectures attendance	Comp Architectures grade	r = 0.336**
		p = 0.001
		n = 109

Networking attendance	Networking grade	r = 0.753**
		p = 0.001
		n = 44

Digi Tech Attendance	Digi Tech grade	r = 0.058
		p = 0.585
		n = 92

User Exp attendance	User Exp. grade	r = 0.601**
		p = 0.001
		n = 93

Programming attendance	Programming grade	r = 0.473**
		p = 0.001
		n = 120

App Prog attendance	App Prog grade	r = 0.388**
		p = 0.001
		n = 87

Notes: (1) n for Networking is abnormally low due to incomplete attendance records for certain classes. (2) ** indicates that the relationship is significant at 0.01 (2-tailed).

5.2.2 Comparison of relationships between L4 and L6 course grades

Table 5.3 indicates strong and highly significant correlations between the grades achieved by students for all their L4 courses. This includes strong relationships between student grades for pairs of related Semester 1 and Semester 2 courses (Computer Architectures & Networking; Digital Technologies & User Experience; and Programming Concepts & Application Programming). There are also strong relationships between courses that have different emphases, in particular, Networking and User Experience (r = 0.774). The strength of these relationships indicates that programming performance is strongly related to performance in other L4 courses. The performance of L4 courses is also strongly affiliated with attendance of these courses.

Table 5.3. Pearson's Correlation analysis between grades for L4 courses.

L4 Grades	Comp Arc (Sem 1)	Networking (Sem 2)	Digi Tech (Sem 1)	User Exp (Sem 2)	Prog Concepts (Sem 1)	App Prog (Sem 2)
Comp Arc	1	0.600	0.545	0.591	0.645	0.548
Networking	0.600	1	0.520	0.774	0.568	0.421
Digi Tech	0.545	0.520	1	0.654	0.622	0.382
User Exp	0.591	0.774	0.654	1	0.693	0.583
Prog Concepts	0.645	0.568	0.622	0.693	1	0.660
App Prog	0.548	0.660	0.382	0.583	0.660	1

Notes: (1) n for Computer Architectures & Networking, 111, Digital Technologies & User Experience, 93, Programming Concepts, 120, Application Programming, 87; (2) all relationships significant at 0.01 (2-tailed).

Similarly, Pearson's Correlation analysis indicates very strong relationships between grades for L6 courses (Table 5.4). These relationships are all significant at the $p < 0.01$ confidence interval. There is a particularly strong relationship between GAD and AIP (r = 0.719) which is most likely because both courses teach and assess students using the C++ programming language.

Table 5.4. Pearson's Correlation analysis of Sri Lankan L6 grades.

	Advanced Programming	Software Engineering	Graphical Programming	Project
Adv Prog	1	0.612	0.719	0.621
Soft Eng	0.612	1	0.611	0.651
Graphical Prog	0.719	0.611	1	0.506
Project	0.621	0.651	0.506	1

Notes: (1) n for all categories: 62; (2) all relationships significant at 0.01 (2-tailed).

Pearson's Correlation analysis also reveals a strong and highly significant relationship between Bucks students' L6 Advanced Programming grade and L6 Project grade (r = 0.741, $p > 0.001$, n = 28). This is further evidence to suggest that learners adopt a similar approach to all L4 and L6 courses.

5.2.3 Comparison of performance in different L6 assessment modes

The L6 advanced programming exam papers tend to contain between six and eight constructed response (CR) questions and no multiple choice (MC) questions. Learners are required to write solutions for various programming tasks that may involve, for example, implementing class diagrams, identifying errors in code fragments and modifying code fragments.

Table 5.5 demonstrates that the average L6 Advanced Programming exam grade (55.74%) is 10% less than the average assignment grade (65.8%). The standard deviation value for exam grades (25.858) is greater than the standard deviation for assignment grades (13.282). This indicates that the exam grades will be distributed more sparsely around the average grade (greater variation), whereas the range of assignment grades will be closer to the average (less variation). The average programming assignment grade of 65.8% for L6 Sri Lankan students is similar to the average programming grade of 63.4% for the 120 L4 Bucks participants, however, both of these are greater than the average programming grade of 56 for the 42 L6 Bucks Advanced Programming students.

Table 5.5. Comparison of average L6 Advanced Programming exam and assignment grade.

	Adv Prog Assignment	Adv Prog Exam
Mean	65.8	55.74
N	62	62
Std. Deviation	13.282	25.858

Table 5.6 and Table 5.7 demonstrate that the average assignment grade was respectively higher than the corresponding exam grade for the graphical programming course (66.89% versus 58.40%) and the software engineering course (56.49% versus 50.77%). Standard deviation values for both subjects are greater for the exam grades, which indicates greater variation in student performance for an examination in comparison to an assignment.

Table 5.6. Comparison of average L6 Graphical programming exam and assignment grade.

	Graphical Prog Assignment grade	Graphical Prog Exam grade
Mean	66.89	58.40
N	62	60
Std. Deviation	10.441	30.850

Table 5.7. Comparison of average L6 Software Engineering exam and assignment grade.

	Soft Eng Assignment grade	Soft Eng Exam grade
Mean	56.49	50.77
N	63	61
Std. Deviation	12.660	24.797

5.2.4 Investigation of individuals who retook courses

Within the parameters of course regulations, students who fail a course at the first attempt normally get a second chance to attempt to course assessment, commonly known as a retake, reassessment or referral. Pearson's Correlation analysis revealed a very strong and highly significant negative relationship (r = -0.703, p < 0.001, n = 120) between students' programming grade and the number of courses they retook in their first year (L4) of undergraduate study. This suggests that students who achieved low programming grades tended to also demonstrate poor performance across several courses and were required to retake L4 courses. Additional Pearson's Correlation analysis (Table 5.8) between the retake count and the grade of related L4 courses also demonstrates strong and negative correlations. This triangulates the findings of previous analyses (Table 5.3 & Table 5.4), suggesting that students perform consistently across all L4 and L6 courses.

Table 5.8. Pearson's Correlation analysis for L4 grades and Retake count.

L4 Grades	Retake Count
Computer Architectures (Sem 1)	-0.622
Networking (Sem 2)	-0.675
Digital Technologies (Sem 1)	-0.661
User Experience (Sem 2)	-0.797
Programming Concepts (Sem 1)	-0.703
Application Programming (Sem 2)	-0.567

Notes: (1) n for Computer Architectures & Networking, 111, Digital Technologies & User Experience, 93, Programming Concepts, 120, Application Programming, 87; (2) all relationships significant at 0.01 (2-tailed).

Further investigation revealed that the students who retook the most courses (five or more) demonstrated poor attendance at L4 programming and most related L4 courses. Pearson's Correlation analysis confirms that a significant negative correlation (r = -0.542, p < 120, n = 120) exists between their attendance of the introductory programming course and the number of L4 courses retaken. Several students missed the first teaching week and some were absent for several consecutive weeks. Authors have discovered that absence in the initial weeks of a programming course tends to disadvantage students due to the sequential nature in which concepts are taught (Ahadi & Lister 2013: 123; Robins 2010).

Categories such as gender and qualification frequently contained similar median scores, albeit differing ranges. However, Figure 5.5 illustrates the presence of outliers in several entry modes and qualifications. For example, despite most BTEC IT entrants not retaking any courses, there were some who retook seven or eight courses. Further investigation found that these individuals had entered with 360 UCAS points. However, external factors, unrelated to academic ability, were known to affect one individual's ability to study.

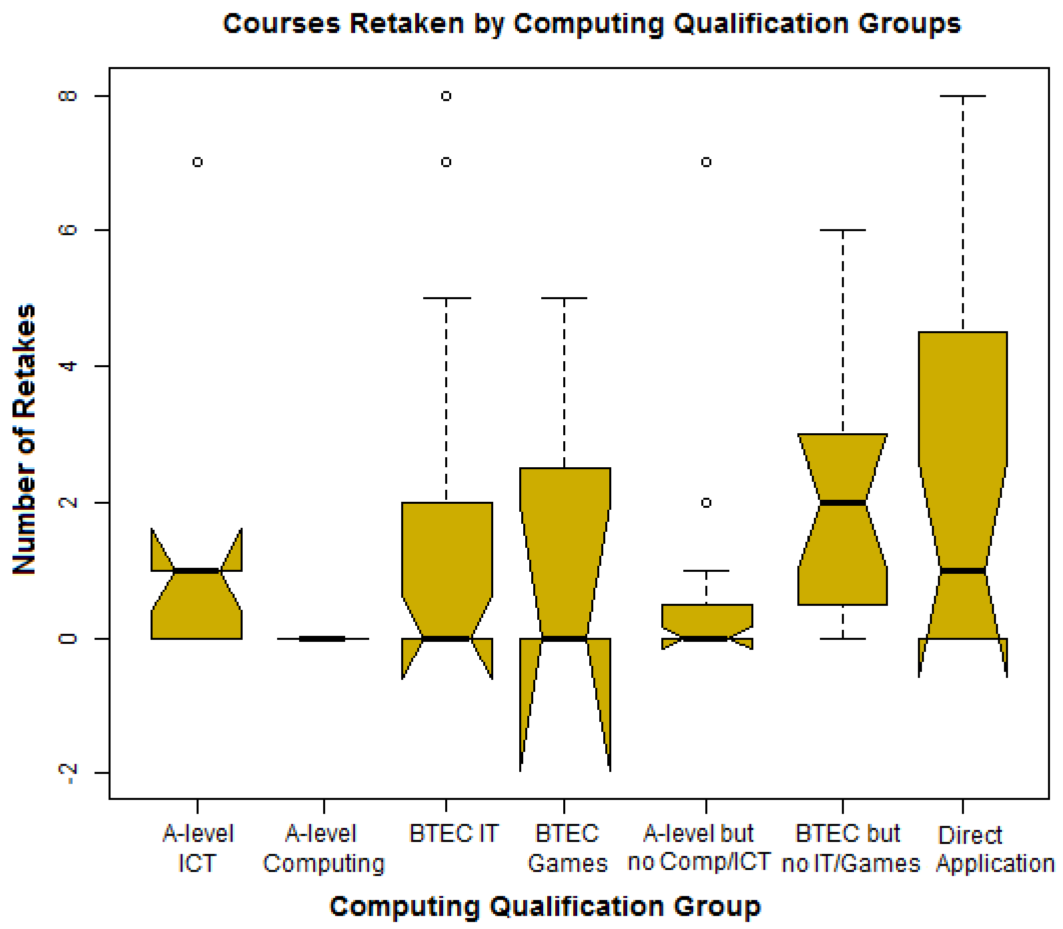


Figure 5.5. Notched boxplot comparisons for the number of course retaken by computing qualification groups. Notes: (1) n for categories is: A-level ICT, 7, A-level Computing, 3; BTEC IT, 26; BTEC Games, 4; A-levels without Computing/IT, 30; BTECs without IT/Games, 16 & Direct Application, 18;

Prior investigation in Section 4.2.1 found that some of the African students who performed poorly were mature students. Continued research discovered that mature students retook more L4 courses than younger students. Some are aged between 25 and 30, and two were between 40 and 45. These mature students also tended to apply directly to the university (not through the UCAS system).

One mature student who had retaken several L4 courses agreed to participate in an interview. He attributed the failure to pass L4 courses first time round to the difficulty in balancing academic commitments alongside work and family commitments. He revealed he did not have access to digital technology when growing up in Nigeria during the 1980s and therefore was not as experienced as his younger peers. However, he was confident that he could learn the programming concepts given enough time; he rejected the idea that they were ‘too hard’ to learn. This finding is consistent with other research (Gordon 2016: 6; MillionPlus 2018; OFFA & OU 2017) that recommends additional support and flexible working arrangements for mature learners.

Summary of Section 5.2

This study has investigated whether trends exist between numerical measurements of course engagement and programming grade for the Bucks and Sri Lanka datasets analysed in Chapter 4.

Consistent with literature (Dolnicar et al. 2009), Figure 5.3 illustrates that attendance typically declines over the duration of the L4 programming course. However, Pearson's correlation analysis demonstrates there is a moderately strong correlation between overall attendance and L4 grade. This corroborates the findings of Section 5.1. Exercises completed in class formed part of the assessment for some cohorts in the sample; however, this explanation would not apply to independent learners or those cohorts which were not required to submit their in-class exercises. Section 5.3 considers the influence of collaborative and interactive behaviours, and Chapter 6 monitors the change in attendance as being a result of course modifications introduced after 2015/16.

Pearson's correlation analysis also demonstrates strong and significant correlations between the grade of related L4 courses and L6 courses (Table 5.3 & Table 5.4). The performance of students in programming is typically consistent with their performance in other courses. Section 5.1 demonstrated that learners can pass without remembering basic programming knowledge. These findings appear to corroborate the conclusion of Chapter 4; that general studying and learning behaviours appear to be a more accurate predictor than specific programming related knowledge or understanding.

Moreover, retake count has a strong negative correlation with L4 programming grade: students who experienced a relatively large number of subject retakes tended to have a lower programming grade. Whilst most of the fail and retake students are mature learners (as previously identified in Chapter 4), some young learners also fail. Both types of student commonly attribute their poor performance to personal issues affecting study or poor time management rather than academic factors (e.g. finding the assignment too hard). Very few Sri Lankan L6 students retook courses. However, as previously stated in Chapter 4, this may be due to the requirement to pass L4 and L5 first, allowing those students to consolidate their skills and learning ahead of taking L6 courses. Section 6.1 further explores the reasons for failure and withdrawal at L4.

Table 5.5 illustrates that the L6 programming exam grade was on average 10% lower (a whole grade) than the average assignment grade. Standard deviation was also significantly greater for exam grade than for assignment grade in all L6 courses (Table 5.6 & Table 5.7). This variation may indicate a wider range of understanding not represented by assignments. The impact of modifying L4 assignment is discussed further in Chapter 6.

5.3 Investigation of collaborative behaviours

This section reports the ethnographical audio/visual study of collaborative behaviours of students learning to program. The motivation for conducting this study is provided by observations of students frequently working together to complete exercises. Moreover, following on from Section 5.1, this study aims to investigate the impact of peer collaboration on programming grade and indicators of code understanding and code knowledge. The purpose of this study is not to evaluate the Pair Programming method as previous studies have (Brought et al. 2010; Maguire et al. 2014; Williams 2007), but rather to record and analyse collaborative behaviours found in the classroom.

Methods

Although this study set out to adopt Teague's (2011) recommendations of recording the same students over a semester to monitor how they change, early recruitment attempts proved that students were unwilling to commit to this schedule. Other reasons included a lack of confidence in their programming abilities to be recorded, and a lack of willingness to engage with optional activities that would not contribute towards course grades. Therefore, a new design of recording standalone episodes was adopted. This study was originally proposed at the same time as the study reported in Chapter 4, and approved by the same internal ethics panel (Appendix 3). Having received key information about the study (Appendix 17) five pairs of students agreed to participate and signed a Consent Form (Appendix 18). Three pairs already had an established relationship, but the other two pairs were working together for the first time.

The researcher instructed students to verbalise their thought processes for the recording but did not provide other guidance beyond typical instructions given to a class. The first pair were provided with a task specially devised for this study (see Appendix 19 for the task description). However, they spent longer than anticipated on the task due to their interpretation of requirements. The study was, therefore, revised to utilise exercises that students would typically work on during practical sessions. Class exercises were smaller in scope and accompanied by guidance and code fragments to structure solutions, thus reducing the potential for misinterpretation and unnecessary complication.

Positioning of equipment

Pair Programming episodes were recorded during normal teaching sessions by a combination of a small handheld microphone and a High Definition (HD) camera (Figure 5.6). The handheld microphone was placed in front of participants, pointing towards their faces. This captured their dialogue more clearly than the camera microphone, which was pointing towards the back of participants' heads.

The camera was positioned such that participants' heads and computer monitor were in frame. Participants were more comfortable with this arrangement than the configuration utilised in commercial studies where the camera is positioned in front of participants' faces. As several pairs spontaneously decided to use laptops or other terminals in recording sessions, positioning the camera behind participants also captured this. Other students working within close proximity to the recording were offered the chance to move elsewhere. All, however, were comfortable to remain seated providing they were not in camera view.

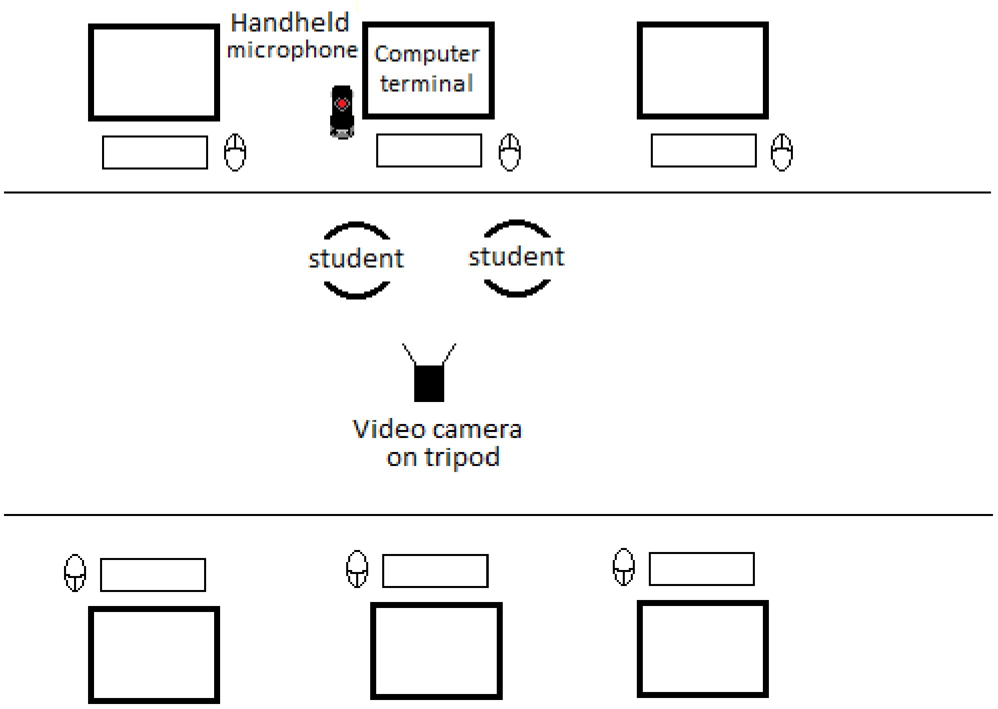


Figure 5.6. The position of video recording equipment in the laboratory.

<u>Analysis technique</u>

Zarb and Hughes (2012) utilised a two-stage approach for their analysis of audio and video recordings. This allowed them to identify key points of interest from recordings before analysing them in more detail. The researcher can then focus on the parts of recordings that reveal underlying motives and learning strategies. Their two-stage process, adopted here, is described below:

Stage 1) The first stage is one of preliminary analysis undertaken by watching videos and "noting down gestures or events that are thought to be of importance" (Zarb & Hughes 2012: 2). This approach is recommended by Wetherell and co-researchers (2001: 36) who suggested that working from recordings (audio or visual) is a good alternative to transcription, "especially for preliminary analysis." Given the potential volume of data, such early selection of the most relevant parts of recordings may significantly reduce the cost and effort of transcription and data analysis.

Stage 2) During the second stage, the relevant sections of the recordings are transcribed, qualitatively interpreted and further analysed. As well as points of interest from individual recordings, patterns may emerge across multiple recordings. These points were then discussed in conjunction with other observations made by tutors outside of this study.

Although Mather (2004) did not use this two-stage analysis process, he included 'fragments' of transcribed dialogue and interaction. This presentation technique is adopted below, alongside any interpretations. Mather (2004) followed Jirotka's (1998) recommendations for presenting transcribed parts of audio/visual recordings. Discussion also refers to non-video recorded observations for the purposes of validating or contrasting behaviours evident in video and audio recordings.

5.3.1 Preliminary analysis (Stage 1)

Preliminary screening of video recordings and audio recordings revealed the following points of interest and patterns that were particularly apparent:

- At several points students attempted to explain corrections to code, but had difficulty in effectively communicating solutions. Some students resorted to taking control of the keyboard from their partner in order to type in the code they were trying to explain. This pattern is discussed in further detail in Section 1 of Stage 2 analysis.
- All five pairs spent little or no time planning or designing their solution before they started coding. This tends to be a common trend in the classes that the researcher has taught or observed at two universities. Some less experienced students have been observed to write ideas down and plan their algorithms on paper, but this is unusual.
- Two pairs (S5 & S6 and S7 & S8) adopted notably different strategies to solve the same exercise. It appeared that students tended to be motivated by program completion but not necessarily interested in understanding the exercise or reflecting on what they have learnt during the process. S7 and S8 were unsure of what they were being asked to do throughout the task but happened to arrive at the correct solution by guessing. Their approaches are contrasted in Section 2 of the Stage 2 analysis.
- The researcher did not assign roles from the Pair Programming (PP) method, but in most cases, participants assumed or arranged roles typical of those specified with PP methods, such as 'navigator' and 'driver' (Beck 2000). There did not appear to be a pattern concerning role allocation. In some cases, the more confident programmer accepts the 'navigator' role,

and in other cases, confident programmers take the ‘driver’ role. Even though the students who agreed to the ‘driver’ role would take their direction mainly from the ‘navigator’, they still contributed to the discussion regarding the direction the pair should take.

- Students sometimes delegated tasks to work on independently, for faster task completion. It was expected that pairs who had not previously worked together might be more reserved in their communication. However, the level of communication between pairs appeared to depend more on their personality rather than how well they knew each other. That being said, the pairs who had worked together previously were more jovial than the pairs that had not worked together before the study. Student roles and communication are discussed further in Section 3 of the Stage 2 analysis.
- One recording revealed how confusing Integrated Development Environment (IDE) error messages were to students. Similar difficulties in error message interpretation are frequently observed by other tutors. This phenomenon is explored further in Section 4 of Stage 2 analysis.

Observations similar to the above are also reported by Zarb and Hughes (2012: 3) in their initial video analyses of recordings of pairs of expert programmers, all of whom were industry professionals. Such similarity is noteworthy given that, with one exception, all subjects involved in the study reported here were novice programmers.

5.3.2 Detailed analysis (Stage 2)

<u>(1) Difficulty in explaining issues</u>

S9 and S10 were tasked with recreating the Rock Paper Scissors game (described as ‘Scissors Paper Stone’ in this context) in a C# Windows Form Application. The pair created and assigned integer values for the player’s choice of item (Scissors = 1, Paper = 2 and Stone = 3) and a method which would randomly generate the computer’s choice (also stored as integer values). They designed a method to compare the two items and generate a result message based on the traditional rules of game (scissors defeats paper; stone defeats scissors; and paper defeats stone). However, they observed that their tests continued to output the same result – ‘user wins’ – despite the player (user) losing. Transcript Fragment 5.1 illustrates S9 struggling to explain why the code was producing this output.

Fragment 5.1 (at 42 minutes 10 seconds recording of video 5):

S9: 'Ah do you know why this isn't working?'
S10: 'Why?'
S9: 'Just because it's three, {turning to look at S10} it doesn't it's always… um…'
{S9 then sighs and hits the desk softly, reclining into his chair, turning to R1 and apologises}
S9 (to R1): 'Sorry'
R1: 'don't worry, don't worry'
{S9 leans forward to point to code}
S9: 'So look, just because.. look, let me show you, look… for instance, this is stone, right, just because it's three… doesn't mean it always beats… how do I explain this?'
S9: 'So pap- just because its three, it doesn't mean… stone beats whatever the user always beats'
S10: 'I know, yeah'
S9: 'So we have- for every- {S9 buries head in hands} 'So I was wrong basically'
{jovially} S10: 'You're always wrong'
S9: 'For every instance of… um…different- {S9 sighs}
S9: 'So instead of having like…' {S9 sighs}
S9: {S9 sighs} 'So how do I do this then… I'm guessing like… we have to have it like this, so, ah, I see…'
S10: 'You know now…'
S9: 'So instead of like having... like, ii-if the userchoice is bigger than compchoice, you need to have… if userchoice equals three {Turning to S10} and compchoice equals… one, and depending on whatever the consequences…'

S9 started trying to generalise: *'just because it's three, it doesn't always mean…'*, but after several incomplete attempts, he then referenced a specific case (concrete example): '*if userChoice equals three, and compChoice equals one*'. With this momentum, S9 asks S10 to draw a table to calculate the winner of the remaining cases. Figure 5.7 illustrates S9 initiating a three by three grid and Figure 5.8 demonstrates the completion of this table.

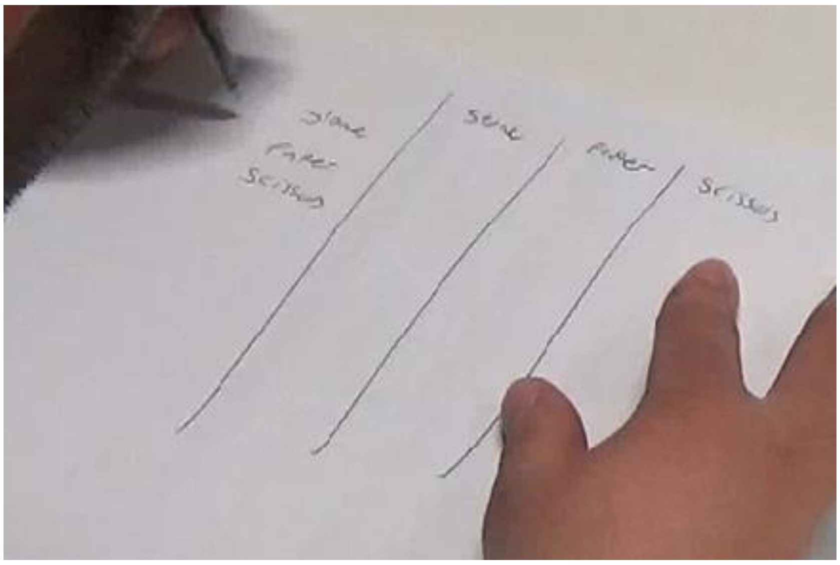
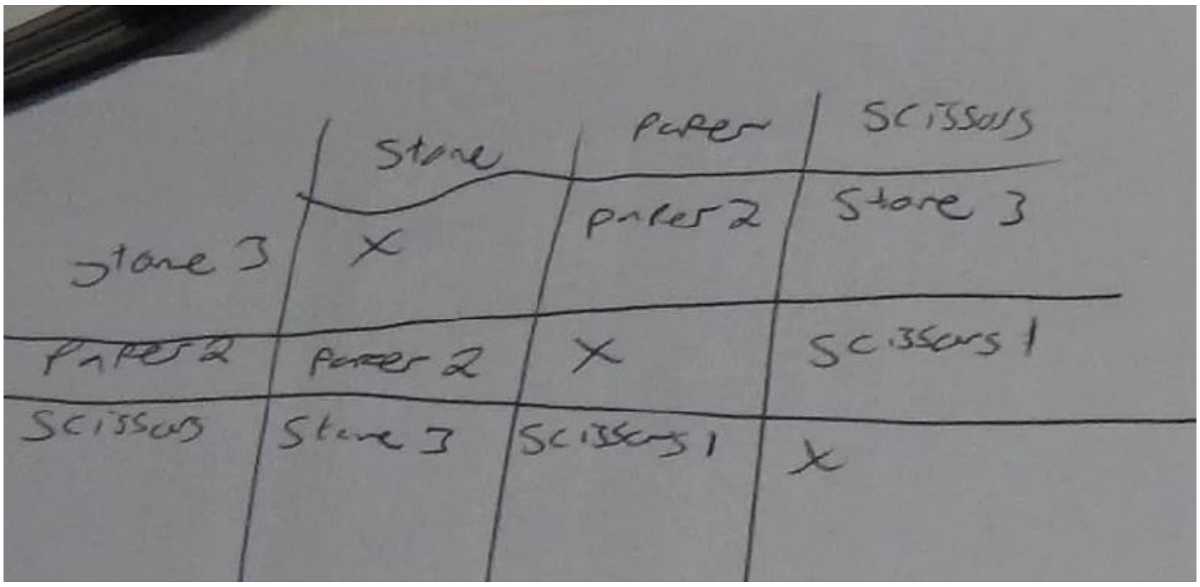


Figure 5.7. Still image from video 5 at 43 minutes illustrating S10 beginning the table of outcomes.



	Stone	Paper	Scissors
Stone 3	X	paper 2	Stone 3
Paper 2	Paper 2	X	Scissors 1
Scissors	Stone 3	Scissors 1	X

Figure 5.8. Still image from video 5 at 45 minutes demonstrating the completed table of outcomes.

After establishing the correct outcomes for game options, through aid of the sketched table, the pair immediately progressed onto the remaining tasks. After the pair completed the exercise and the recording ceased, S9 commented that the table helped him to clarify what he was trying to communicate to S10 verbally. Although S9 did not articulate this newfound understanding at any point, he acknowledged that he '*was wrong*' at the time, but neither he nor S10 verbalised why they were wrong or tried to discuss the issue further.

In another recording with a different pair, S6 also experienced difficulty when articulating his thoughts. This exercise required learners to program their robot to shoot at surrounding enemy robots (AlienSpiders). Participants had to use methods to get the angle required to turn and face the enemy spider, and the distance between their robot and the enemy spider. The distance could then be used to look up the angle required to aim the cannon (vertically) to destroy the enemy spider. Fragment 5.2 demonstrates S6 taking the keyboard from S5 in order to solve an error.

Fragment 5.2 (at 3 minutes 45 seconds of video 3):

S6: 'Ah, I get it… right, so…'
{S6 takes control of the keyboard from S5}
S6: 'What you want to do- so we've got the direction now, so it's been turn- and now we want the distance, because the distance has already allocated, all the way up to 63, and the furthest- so you can now get the distance… so dis equals…'
{S6 stops speaking and starts coding silently, with S5 watching the screen.}
S6: 'Now that's going to get the distance, now we should be able to… um…'
{S6 resumes coding silently}

Fragment 5.3 illustrates S6 returning the keyboard to S5, and dialogue resumes as S6 attempts to explain his code to S5.

Fragment 5.3 (at 5 minutes 30 seconds of video 3):

S6: 'Try that.'
{S6 then gives the keyboard back to S5}
S5 {looking at the code}: 'What about… it shoots in angles though'
S6: 'Yeah… yeah, I've done that, look.'
S6: 'Right so what this is going to do-'
S5: 'That's just saying 'fire(1)'
S6: 'Not that, you prune! Here...'
{S6 points to the statement on screen}
S5: 'Oh right' {S5 laughs}
S6: 'It's going aim at the angle-
S5: 'Yeah' {laughing}
S6: 'At the angle of the distance, so here, so say, the distance is 60, its gonna aim at angle 60, which is 35.17'
S5: 'Alright… alright.'

The actions of S6 taking control of the keyboard and S9 asking S10 to draw a table are interpreted to be alternative supporting strategies to extend beyond the simple communication of their ideas verbally. S9 experienced the most difficulty with explaining his idea verbally, and this behaviour appears to indicate a preference to run the code and confirm it works as expected before attempting to explain ideas. The code also provides example values (concrete examples) to refer to, as did S9.

The formation of mental models of abstract concepts was discovered to correlate with programming success (Allert 2004). However, Kolb's (1985) learning cycle posits that mental models are developed from concrete examples (in this case, observing how a program responds to code). Similarly, Teague and colleagues (2013) theorise that tracing code (reading and describing each statement correctly) is a prerequisite to forming abstractions (mental models). The authors went on to adapt the stages of Piagetian theory (Piaget 1952) for the development of programming skills.

A dependence on visual prompts (concrete examples) such as tables and reference to values specific to the task is characteristic of the pre-operational stage (Teague 2015: 265). S6 and S9 demonstrated some proficiency in tracing code (sensorimotor stage) but they may require more experience or examples to consistently interpret code behaviour (pre-operational stage) in order to identify underlying patterns which then inform mental model abstractions and enable reasoning without concrete examples or values (operational stage) (Teague et al. 2013).

S1 also made a distinction between understanding how programming concepts are applied, and the skills to communicate them. Despite his years of professional experience, he recognised that he did not have the communication skills to teach some concepts effectively.

<u>(2) Strategy comparison</u>

S7 and S8 attempted the same class exercise as S5 and S6, as described in the previous section. However, the pairs adopted quite different approaches to solve the task.

This is first illustrated in the level of preparation that each pair adopts. S8 opens the task almost immediately after recording starts but he did not expect to see pre-written code. After thinking it might be an error or code belonging to another student, S8 restarts the program but the code remains. On the other hand, S5 and S6 expect the pre-written code because they started by reading the task description aloud and even checked the code was there as they were reading. Interestingly, S8 suggests that he and S7 should read the task description together, but S7 appears to indicate this is unnecessary by responding '*nah… easy, kill them all…*'. Despite this, S8 takes roughly a minute to read the task description by himself. He accepts the provided code, but later makes the comment: '*I don't get it, why do we have 64 arrays, of different angles?*' This indicates that arrays might be misunderstood - as the code supplied is one array of 64 elements, rather than 64 arrays. S8 goes on to enter code supplied by task description although he admits to S7 that he is unsure what it means. However, the code provided is only part of the solution. Therefore, when the pair executed this code, the robot did not move. The pair agree to 'step through' each line of code in the solution to learn more, but they fail to recognise that there are missing statements. In contrast, S5 and S6 appear to demonstrate better understanding of task requirements, and implement a more organised approach to problem-solving. They built their solution incrementally; first coding the robot to shoot one AlienSpider (partly illustrated in Fragment 5.2 & Fragment 5.3); then they repeated this code (by placing the code in a loop) to shoot the remaining spiders and complete the task.

S7 and S8 appear to be adopt more of a 'trial and error' approach to problem-solving. Further comments by S8 indicate that this task is not fully understood: '*what are we supposed to be doing?*' S7 does not answer, but instead they both read the task description for approximately 30 seconds. S8 tentatively suggests '*okay, I guess we're going to shoot them'.* S7 suggests repeating the statements 12 times, before hastily suggesting 24 repetitions. S8 amends the code and they find that the code successfully destroys all the spiders, completing the task. However, despite completing the task, Fragment 5.4 demonstrates that the programmers still do not fully understand their solution.

Fragment 5.4 (at 7 minutes 10 seconds of video 4):

> *S8: 'funnily enough it works, but I have no idea what I've done'*
> *S7: 'that's how I completed a couple of tasks.'*
> *{ both S8 and S7 laugh }*
> *R1: 'Done?'*
> *S8: 'I'm not sure if we have or not-'*
> *S7: 'forget about the booklet!'*

With the requirements satisfied, S7 and S8 revisit the code to further improve its efficiency. S8 first exchanges their defined 'for' loop for an infinite 'while' loop due to uncertainty regarding the number of iterations. He then realises that a dynamic 'do while' loop which repeats until enemy spiders are no longer detected is more efficient than his first revision.

This post-completion discussion is interesting for two reasons. The first is timing. The pair was comfortable experimenting once they had developed a working solution. Analysis of the video recording demonstrates that little time was given to planning or discussing their approach during the problem-solving process. Even though S8 spent most of the task stating that he did not understand the purpose of the task, he later demonstrated confidence in understanding the difference between the three types of iterative statements and determined the most suitable statement for their context. The second observation is that this discussion is unique; none of the other pairs evaluated or discussed their work after fulfilling the requirements. Most were keen to cease recording so they could progress with other tasks. Even though S5 and S6 did invest time in deciding the best approach before coding, they did not pause to reflect or discuss any improvements. Both these observations suggest that task completion is a more prominent motivation than the discussion of the underlying principles of programming. In one non-video recorded observation, one student made his task completion motive explicit: '*Who cares how it works!? As I long as I get the marks, that's all I care about.*' This is consistent with the contrasting motivations discussed in literature (Elliot et al. 1999; Porter & Zingaro 2016). Whilst authors noted that students tend to alternate between surface and deep approaches (which tend to correspond with performance and mastery motives, respectively), S7 and S8 demonstrate that learners may be more inclined to discuss their approach once they have satisfied their objective.

The pairs also adopted different approaches towards solving the task. Despite S5 and S6 investing a greater amount of time than S7 and S8 in preparation by reading the task description and initial discussion regarding requirements, S8 refers to the instructions on multiple occasions but still seemed unsure as to what was required. Appendix 21 contains the task description, and the aim is clearly stated under the heading *Your Task,* which is to '*destroy all AlienSpiders'*. S7 acknowledges this at the start of the recording ('*easy, kill all of them'*) but does not respond to S8's later questions regarding '*what are we supposed to be doing?*' nor does he refer to the task description at any point.

However, S7's comment: '*that's why I never follow the booklet'* suggests that he does not trust the instructions or deem them to be helpful. Whilst S8 eventually, and somewhat indecisively, states that the aim is to destroy the spiders, both he and S7 lack a strategy of how to do so. On the other hand, both S5 and S6 regularly consult the task instructions, but seem to have a clearer idea of how to design their solution. The recording illustrates that their approach is more organised. They applied an incremental strategy of coding the steps to destroy one spider, then repeated those steps to destroy the remaining spiders.

The second bullet point in Appendix 21 does outline the sequence of steps required for the solution; however, the first bullet point attempts to explain the significance of the lookup table. S8's comment: '*I don't get it, why do we have 64 arrays, of different angles?*' suggests that he does not understand that it is one array of 64 elements, and the repeated uncertainty about the task suggests he does not see the purpose of the table.

This confusion with the first bullet point may have been a source of cognitive overload (Sweller et al. 2011) and prevented S8 from concentrating on the second bullet point. It is possible that novices, who require structured and explicit guidance due to their primitive levels of cognition (Bennedsen & Caspersen 2005; Teague et al. 2013), may benefit from being presented with a design of the intended solution, before attempting to explain the significance of individual steps. Therefore, a minor change such as reversing the order of the bullet points (1) and (2) in Appendix 21 may enable novices to comprehend the purpose of the task, and recognise how the lookup table relates to the purpose of the task. Given that novices are usually operating at the initial cognitive stages, they may not be able to translate a high-level task description to code, especially as individual steps are not provided. Therefore, provision of full or partial algorithms to translate may be initially more appropriate for novices. Then gradual reduction - similar to the 'fading' principles described in Worked Examples (Skudder & Luxton-Reily 2014) – may enable them to transition more effectively towards higher-level descriptions.

(3) Pair Programming roles and communication

As previously stated, all pairs assumed roles similar to the 'driver' and 'navigator' roles in the Pair Programming technique despite not being told to do so. The purpose of this study was to capture and learn more about typical classroom behaviours; therefore, pairs were also not instructed to switch roles every 12 to 20 minutes as required by some educational studies (Brought et al. 2010; Wood et al. 2013). Analysis of video recordings demonstrated that the more experienced and/or confident programmers tended to direct or allocate responsibilities. However, not all the experienced programmers became the 'navigator'. For example, S9 was more vocal than S10 yet he asked S9 to 'navigate' him. Whereas, S1 – the programmer with professional experience – took navigating responsibilities, asking S2 to code.

Interestingly, some experienced participants initially directed their less experienced peers but became the ‘driver’ later in the recording. Fragment 5.2 earlier illustrated S6 taking control of the keyboard to code his idea. However, S6 did return possession of the keyboard to S5 once finishing. On the other hand, S1 takes control of coding responsibilities for the remainder of the session. Transcript Fragment 5.5 illustrates S1 beginning to work independently on his own laptop to resolve an error. After S1 begins to advance the code he copied, S2 asks S1 if he wants to copy the code across to the computer they started working on, but S1 advises that they should continue on his laptop as it is currently working on there. S2 then watches S1 take ownership of coding responsibilities (‘driving’) until they complete the task, whilst making suggestions.

Fragment 5.5 (6 minutes 42 seconds recording of video 1):

{S1 and S2 are reading the task description…}
S2 {reading aloud}: ‘it should display the category of each robot…’
S1: ‘Actually, it does tell us- there… {S1 uses his pen to tap on the relevant sentence on the description} we have our six types, so we can put them straight in the array.’
{S1 checks F1 on his laptop (out of sight) and finds how to store items in an array}
S1: ‘So under that we can just put ‘Bots[1]…’
{S2 starts typing}
S1: ‘I’ll read the rest of it to see if we’ve missed anything else.’

Whilst S9 and S10 do not switch roles, S9 does take an active role in navigating by reading the booklet with S10 (as did S7 with S8). S9 does also ask S10 to create a table, as previously illustrated in Figure 5.7 and Figure 5.8. Even though S9 is vocal, S10 also uses his time to think through and solve errors, as demonstrated in Fragment 5.6.

Fragment 5.6 (at 23 minutes 24 seconds recording video 5):

{S10 looks back and forth from the booklet and the terminal}
S10 {tentatively}: ‘I… think… I just realised where I-wait… what?’
S10: ‘Why is yours say ‘background image’
S9: ‘what you talking about?’
S10: ‘go on the thing, why is it background image?’
{S9 pauses for a second or two and then lets out an enlightened ‘ahhhhh..’ turning to face S10}
S10: ‘no wonder why, init…that…look this says image, not background image’
S9: ‘that’s why’
S10: that’s why this doesn’t work’

S3 exhibits a temporary change from navigating to actively coding or searching for resources on two occasions. The first time S3 uses his terminal to check the syntax in a program he had already written. He then uses his computer a second time to locate images on a resource hard drive. Having found where the images are located, he then tells S4, and resumes his role of navigating.

The educational value of the working format is diminished when confident programmers take control of the keyboard to complete the task for their less experienced peer – reducing them to observers (Stephens & Rosenberg 2003). Whilst these recordings do demonstrate confident and experienced programmers taking ownership of coding, S5 did get the opportunity to ask S6 to explain his code. S6 did also relinquish control of the keyboard and allowed S5 to code until they completed the task. Whilst S1 did not return control of the keyboard he did pay careful attention to involve S2 and they frequently discussed task issues and direction together.

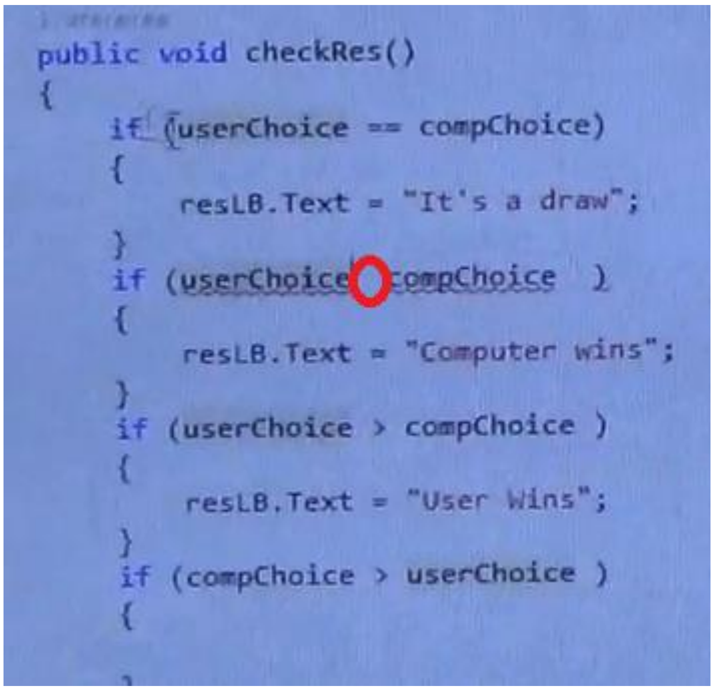
S1 and S2 had not previously worked together, and were probably the pair with the greatest difference in programming ability and experience. S1 has many years of experience working as a tester in industry, whereas S2 is not known to have any previous programming experience. Despite not working together before, they still maintained frequent communication. In addition, S7 and S8 had also no prior relationship when participating in the recording session, and did not appear to have any problems communicating. On the other hand, pairs that did have a prior relationship were more comfortable being jovial and calling each other names without taking offence (as demonstrated in Fragment 5.1 and Fragment 5.3). Conversely, other students with a prior working relationship (S3 & S4) had more reserved personalities and, therefore, communicated less frequently, but when necessary.

Within the published literature mixed results are reported regarding pairings. In some cases, pairing programmers of similar experience and ability was most beneficial (Stephens & Rosenberg 2003); however, in others, pairing of dissimilar programmers produced work to a higher standard (Sfetsos et al. 2009). These recordings demonstrated that those with a prior relationship did not necessarily complete the exercises any quicker or to a higher standard than those who had not worked together previously. This suggests that prior experience is an unreliable metric to distinguish between strong and weak performance.

Interestingly, even though S3 and S4 conversed less than the other four pairs, S3 stated that verbalising his thoughts helped him understand the topics in conversation more clearly. This indicates that dialogue and verbalisation can be effective in enabling deeper understanding. However, instructor intervention may be required to concentrate pair conversation if programming topics are not frequently discussed at length.

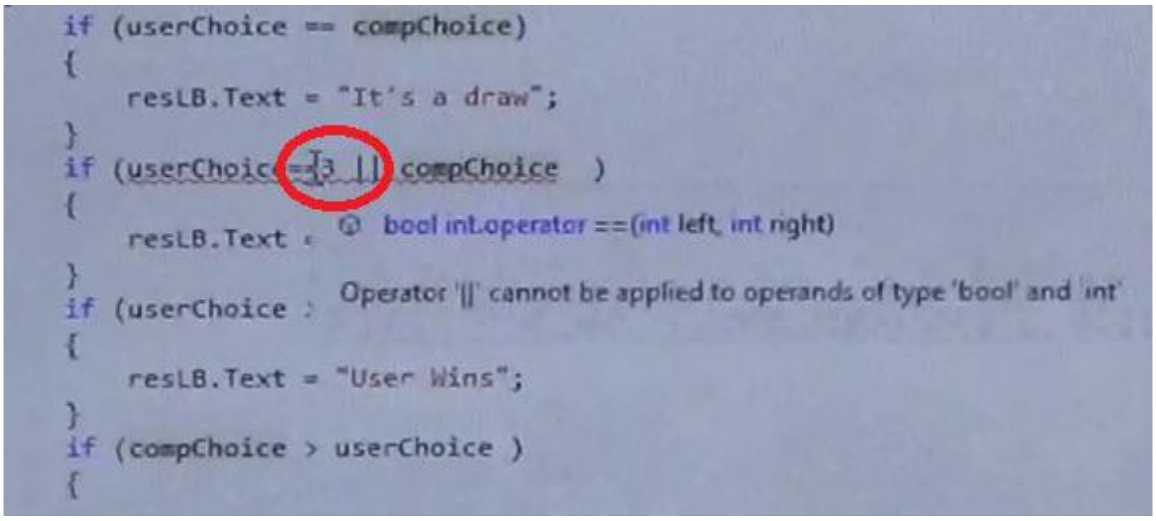
(4) Error messages

S9 and S10 also had difficulty interpreting Visual Studio™ (VS) error messages. Continuing from Figure 5.8, Figure 5.9 illustrates S9 deleting the 'less than' operator (<) between variables 'userChoice' and 'compChoice' in anticipation of comparing each of these with numeric values representing items (Paper = 1, Scissors = 2 and Stone = 3). Despite intending to use the AND operator (&&) but mistakenly writing OR (||), Figure 5.10 illustrates the statement being underlined in red before completing the other comparison.



```
public void checkRes()
{
    if (userChoice == compChoice)
    {
        resLB.Text = "It's a draw";
    }
    if (userChoice compChoice  )
    {
        resLB.Text = "Computer wins";
    }
    if (userChoice > compChoice )
    {
        resLB.Text = "User Wins";
    }
    if (compChoice > userChoice )
    {
```

Figure 5.9. Still image of video 5 at 46 minutes 48 seconds, showing S9 deleting the arithmetic operators from the conditional statements.

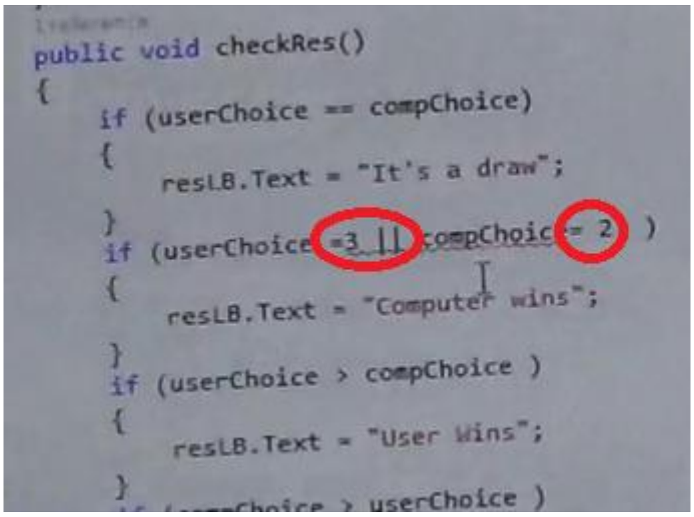


```
if (userChoice == compChoice)
{
    resLB.Text = "It's a draw";
}
if (userChoic==3 || compChoice  )
{
    resLB.Text =    bool int.operator ==(int left, int right)
}
if (userChoice >    Operator '||' cannot be applied to operands of type 'bool' and 'int'
{
    resLB.Text = "User Wins";
}
if (compChoice > userChoice )
{
```

Figure 5.10. Still image of video 5 at 46 minutes 54 seconds, showing S9 attempting to join comparisons with an OR operator.

S9 became agitated by the red underline, but deletes one equal sign (as demonstrated in Figure 5.11), changing the comparison operator (correct for this application) to an assignment operator (incorrect for this context). This did not resolve the error – instead it adds further to the number of errors. But S9 ignored the red line, and completed the second comparison with a single 'equals' sign (Figure 5.11).

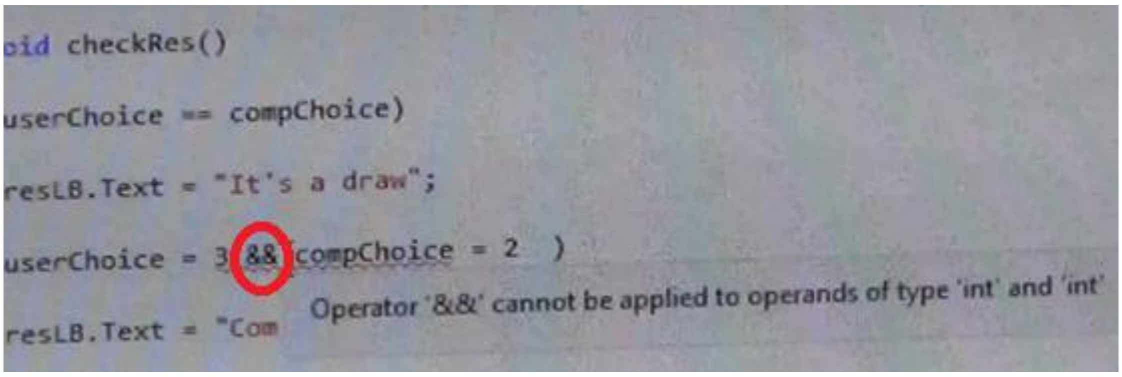
Failure to know the difference between the comparison operator (==) and assignment operator (=), and a keenness to complete the task quickly, meant that S9 opted for a trial and error change to try and perform a 'quick fix'.



```
1 reference
public void checkRes()
{
    if (userChoice == compChoice)
    {
        resLB.Text = "It's a draw";
    }
    if (userChoice =3 || compChoice = 2  )
    {
        resLB.Text = "Computer wins";
    }
    if (userChoice > compChoice )
    {
        resLB.Text = "User Wins";
    }
                  > userChoice )
```

Figure 5.11. Still image of video 5 at 47 minutes 17 seconds, showing the red line underneath the erroneous code.

S9 does realise that he mistakenly wrote the OR operator (||) when he intended to use the AND operator (&&). However, assuming that was the mistake, he is confused when the error still persists (Figure 5.12).



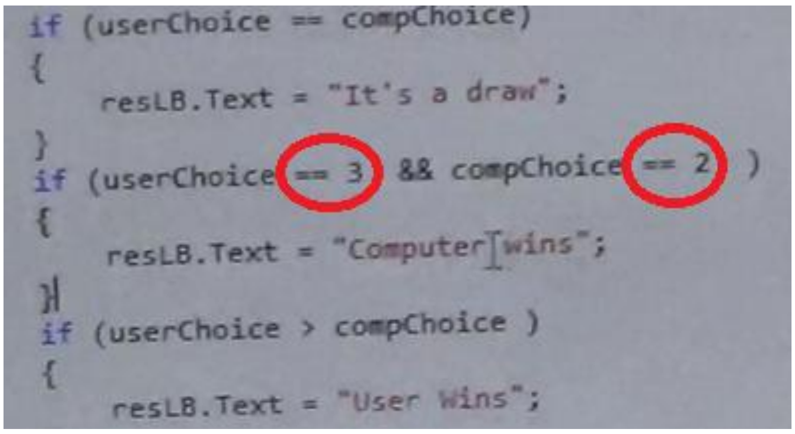
```
oid checkRes()
userChoice == compChoice)
resLB.Text = "It's a draw";
userChoice = 3 && compChoice = 2  )
resLB.Text = "Com   Operator '&&' cannot be applied to operands of type 'int' and 'int'
```

Figure 5.12. Still image of video 5 at 47 minutes 36 seconds, showing the red line underneath the erroneous code.

Transcript Fragment 5.7 illustrates the dialogue between S9 and S10 as they read and attempt to interpret the error. From reading the error message, S10 thought the logical operator was wrong (*Operator AND cannot be used*), however, whilst S9 struggled to articulate the reason, he is convinced that AND is the correct operator. S10 then resolves the error by adding another equals symbol – which might be a guess as no reason or justification is given. This changes the erroneous assignment operators to comparison operators (Figure 5.13).

Fragment 5.7 (at 47 minutes 49 seconds of video 5):

S10 {reading from the screen}: 'Operator AND cannot be applied to operands of types int and int…'
S10: 'is it int?'
{S9 scrolls up the code to check the data type of the variables}
S9 {in disbelief}: 'what…'
{S9 then scrolls back down, taking a few seconds to think}
S9: {reading from the screen}: 'Operator AND cannot be applied to operands of types int and int…'
S9: '…that's really annoying'
S10 {using his pencil to gesture at the two variables}: 'can't be int and int'
S9: 'yeah but.. it doesn't like that this is-'
S10 {said assertively}: 'has to be OR'
S10 {then repeated more softly}: 'it has to be OR'
S9: 'no, 'cos you need to have ii-if…nah, that's right, I don't know why it doesn't like it. Uhmmm…'
S9 {reading from the screen again}: 'Operator AND cannot be applied to operands of types int and int… that's stupid though…'
{S9 then proceeds to change the operator to OR}
S9: 'don't like that either'
{S9 then silently deletes the operator, adds an additional equals sign for comparison on both conditions, and then adds the AND operator in between them. The red line then disappears}
S9: 'there we are'
S10: 'double equals, that's what you need'



```
if (userChoice == compChoice)
{
    resLB.Text = "It's a draw";
}
if (userChoice == 3 && compChoice == 2 )
{
    resLB.Text = "Computer wins";
}
if (userChoice > compChoice )
{
    resLB.Text = "User Wins";
```

Figure 5.13. Still image of video 5 at 48 minutes 43 seconds, showing the correct syntax for combining comparisons with an AND operator.

It is interesting that had S9 not been deterred by the appearance of the red underline in Figure 5.10 and added the second comparison operator and value after ‘compChoice’. The code would have been syntactically correct (the red underline would have disappeared) but logically incorrect (OR operator was mistakenly coded instead of AND). Later test runs of the program would have produced incorrect results, causing further code modifications. However, S10 did realise that he had confused the symbols, so he would have probably resolved this issue. Whilst he was not able to articulate clearly (Fragment 5.7), he was aware of the difference between the operators.

Although novices would be expected to tell the difference between the assignment and comparison operator, few would doubt or question instructions from an IDE. Therefore, an error message stating that the ‘*AND operator cannot be used*’, would be likely to provoke novices to change their operator without question, as demonstrated in Fragment 5.7. Numerous observations outside of this study – multiple institutions with different IDEs - confirm that students are often confused by ambiguous error messages. IDEs often produce a long list of errors, which have been found later to be the result of a missing brace or semicolon.

Conventional IDEs do not allow for customisation of their error messages; however, explanation of typical error messages may help learners interpret them with less frustration and allow for quicker progression. Recently developed IDEs (examples include ‘Rust’ and ‘Elm’), in addition to tools and plugins (such as ‘ReSharper’ for C# and ‘Implicitly’ for Scala), offer more intelligent and relatable interventions for solving errors (Cantero 2018; Plociniczak 2015: 2). Interventions such as these can help reduce the obstructions and frustrations that negatively affect student motivation and momentum (Robins 2010).

5.3.3 Investigation of the collaborative behaviours amongst Games programmers

Complementary to the analysis of video recordings, this section reports additional examples of collaborative and group interactions which may be influential to learning success. Tutors observed that the Independent Games Production (IGP) and Games groups engaged with course material and interacted with each other to a greater extent than other groups. These named groups also outperformed other degree programmes, as illustrated in Table 5.9.

Table 5.9. Comparison of final grade averages for the 2015/16 Introductory Programming course.

Group	Time	Mean grade	Number of students
BA Independent Games Production (IGP)	Monday 9am	61	15
BSc Computing	Monday 1pm	47	15
FdSc Computing	Tuesday 9am	52	12
Mixture of BSc Computing and FdSc Computing	Thursday 1pm	50	15
BSc Software Engineering	Thursday 1pm	48	9
BSc Games Development	Friday 9am	76	17

Notes: (1) The final grades were calculated from an average of the three coursework parts (A, B and C). (2) Students who did not submit work were not included in this average.

The intention was to interview learners from the 2015/16 IGP class and the 2015/16 Games class as groups to investigate typical behaviours that promoted learning success. However, due to the unavailability of those who agreed to participate, it was not possible to conduct group interviews for these two categories. Instead, participants were interviewed separately.

Interviews revealed key behaviours that strengthened the group dynamic and had possibly influenced their programming performance:

- Learners in both groups regularly socialise and develop bonds with each other outside of class in pubs, fast food chains, university events and gaming conventions.
- Both IGP and Games students share a passion for video games and often devote a significant amount of time to playing them.
- Members of both groups regularly communicate via social media. One said that “we created a group chat on Facebook, which we used for talking about coursework and also arranging social stuff”.
- The group members regularly meet up in the computing laboratories during free periods of the day to work together on exercises.

- The more experienced programmers in the two groups taught and provided guidance for their less experienced peers. Interviews revealed that the experienced programmers did not share their code for their peers to copy but instead asked appropriate questions and prompts towards helping their peers solve the problem.
- Several members acknowledged a growth in their confidence and sense of identity during the first year. Most attributed this growth to the friendships made within the group and from belonging to a community. One student called the group her "family".

The strong bonds between the members of the IGP group are illustrated by one occasion when the entire group arrived late to class together. It transpired that the group had arranged to meet up before class, but were late because they were waiting for one member of the group.

The comment that regarded the group to be family characterises the type of belonging that was found to be conducive to success (Thomas 2012a). The more confident and experienced members of the group looked out for and included those less experienced; providing access to resources and individual support. This behaviour embodies the Legitimate Peripheral Participation (LPP) theory (Lave & Wenger 1991) which underpins how 'communities of practice' influence and inspire change through social interaction. One interviewee, who referred to himself as an introvert and as someone who preferred to work on his own, found that his capacity to be around people grew when he started to socialise with other confident members of the group.

Students are less prone to distraction when they are working amongst peers (Avila 2016; McKinney & Denton 2006; Williams 2007). Regular collaborative working sessions outside of class may explain their high levels of achievement. Moreover, the discovery that experienced programmers enable dialogue characteristic of the deep learning approach (Marton & Saljo 1976) – concerned with underlying programming principles and problem-solving – is distinct from the dialogue concerned with task completion in video recordings. The more experienced learners would tailor their teaching and recommendations to the individual needs of their peers. This characterises Vygotsky's Zone of Proximal Development (ZPD) theory which suggests that learning is most effective when new knowledge is an extension of what one already knows (Vygotsky 1976; Wood et al. 1976).

When such bonds do not exist between a group of students, educators can play a key role in facilitating and encouraging connection between learners. Small acts such as connecting like-minded learners, including minority students and contacting absent students to check everything is okay can help learners to increase their sense of belonging and feeling valued. In addition, educators can also provide resources to equip and empower the most confident members to better teach and mentor their less experienced peers.

Summary of Section 5.3

Analysis of video recordings revealed that most programmers struggled to articulate and verbally explain abstract and intangible code-related ideas. Instead, they adopted alternative methods of communication, such as visual aids and coding their ideas. These actions are representative of the initial stages of cognitive development (Kolb 1985; Teague et al. 2013), which tend to concentrate on specific and observable changes to code ('concrete examples'). Regarding alternative communication methods, it was typically the more confident of the two programmers in a pair who took ownership of the keyboard. Despite participants allocating or assuming roles once recording had commenced, most were comfortable with interchanging of these roles for advancement of the solution. Video analysis found no significant difference between the performance of pairs with and without prior working history. The distance between the participants' experience was not interpreted to impact success or dialogue. Instead, the motivations and working approaches were far more influential on outcomes.

This study demonstrates a distinction between dialogue that concentrated on task completion and dialogue that aimed to understand the underlying coding concepts (comprehension). In video recordings, dialogue was primarily concerned with specific task ideas. In one episode, a pair only considered improvements to their code once they had completed the requirements. However, interviews with Games and IGP students revealed dialogue with the intention of developing understanding of coding principles and imparting problem-solving processes. This contrast exemplifies the application of deep and surface approaches (Marton & Saljo 1976), which may stem from underlying mastery or performance-based motives (Zingaro 2016), respectively. Such a distinction may also explain the patterns previously identified in Section 5.1; namely, how collaborative behaviours may aid more efficient completion of exercises, but not necessarily promote deeper understanding of programming concepts.

Participant comments provide some evidence that the verbalisation of thinking encouraged during Pair Programming episodes helped to aid clarification of ideas and develop understanding, in addition to representing typical workplace behaviours. However, further work is required to discover more effective means for directing dialogue towards underlying principles of programming – to encourage deep learning. In addition, concise and intentionally designed exercise descriptions may help reduce cognitive overload (Sweller et al. 2011) and facilitate quicker advancement through cognitive stages.

Both video recordings and anecdotal observations across multiple cohorts at multiple institutions demonstrated that IDE error messages are a common source of confusion and irritation. Novices lack the experience and processes to make informed corrections to their code; causing them to guess and make minor changes until the error disappears. One video episode demonstrated that this trial and error strategy only led to further complications and caused the student programmers to spend an unnecessary amount of time trying to resolve the error. Future intervention from tutors should help learners to interpret error messages and view them as learning opportunities rather than obstacles.

5.4 Chapter 5 Conclusion

Continuing from the investigation in Chapter 4, which sought to determine the value of preparatory qualifications and applicant data in predicting learning success, this chapter addresses Objective 4 "*Explore patterns of engagement with L4 programming learning environments towards identifying potential indicators of success and vulnerability*" as stated in Section 1.5.

The multivariate analysis reported in Section 5.1 revealed a disconnect between L4 programming grade and an end of course measurement of code knowledge and code understanding. Furthermore, the analysis illustrated a misalignment between self-reported measurements of collaborative classroom working habits and indicators of code knowledge and understanding.

Section 5.2 revealed that attendance correlated moderately strongly with L4 grade. This also coincided with strong relationships between the final grade of related L4 and L6 courses. This consistency suggests that students adopt a similar approach to all assessments, for which attendance of taught sessions often directly contributes to assessments.

Video and audio analysis of collaborative classroom behaviours in Section 5.3 revealed that participants tended to be motivated by fulfilling task requirements before task comprehension. Students were observed to adopt trial and error strategies to resolve errors and hasten the completion of tasks, paying less attention to discussion and learning opportunities. Some participants were able to arrive at a working solution without understanding the purpose of the task or their code. Such findings may, in part, explain the disconnect between collaboration and code understanding and knowledge.

The observations were also consistent with cognitive learning theories, such as Neo-Piagetian Theory (Teague et al. 2013) and Kolb’s (1985) Learning Cycle, which proposed that most novices were operating at the initial stages of cognitive development and were not able to demonstrate higher-order thinking skills such as reasoning and explanation of abstract concepts.

Interviews with two groups of highly motivated and cohesive games students revealed their key behaviours behind their success. Not only did their high level of cohesion enable strong bonds and belonging to the group, but their dialogue was distinct from the typical communication more commonly prevalent in video recordings. The most experienced programmers in each group directed their less experienced peers towards understanding concepts and recognising the importance of underlying processes, as opposed to simply meeting exercise requirements. This behaviour resembles the deep investigative approach (Marton & Saljo 1976) also strongly associated with mastery goals (Zingaro 2016).

These findings provide useful guidance for informing teaching interventions. Chapter 6 discusses changes to an introductory programming course and evaluates the impact of these modifications on learning performance. Chapter 6 also recommends a pragmatic teaching model, underpinned by case studies and educational theory, which can be applied in contemporary contexts of HE delivery.

Chapter 6 Evaluation of course modifications and teaching methods

Situation prior to curriculum changes

Prior to the 2015/16 academic year, the introductory programming course for Bucks L4 students was taught via weekly three-hour sessions. These teaching sessions consisted of a one-hour lecture in which programming concepts were taught with aid of a slide presentation. This was then followed by a two-hour practical session in which students attempted exercises corresponding to the concepts taught.

In the first week, 'study packs' were provided containing exercises to be attempted during class practical sessions as well independent exercises to be conducted outside of class. Before the 2014/15 academic year, students had been assessed on their attempts at both the class exercises and independent exercises (numbering some 100 exercises in total). However, starting in 2014/15, this number was reduced to some 40 exercises and only the independent exercises were required to be submitted for assessment. Even though most students completed the unassessed class exercises first, some ignored these and attempted the independent exercises first during the scheduled practical sessions. However, one tutor made a request to reinstate the assessment of class exercises during the 2018/19 academic year to more closely monitor progress and encourage attendance.

The following sections discuss successive changes to the introductory L4 programming course curriculum at Bucks and evaluate the impact of those revisions on student learning. Section 6.1 evaluates changes to attendance, non-submission rate and failure rate. This data is typically available for all tutors when assessing student work, and has been fully anonymised when reported in this chapter. Pearson's correlation analysis is applied to evaluate the usefulness of early formative quiz scores for predicting L4 grade in Section 6.2. Section 6.3 contributes a case study of recent changes to the format of teaching sessions, informed by established learning theories. These are evaluated by t-tests applied to pre- and post- quiz measurements to evaluate trends in learning gain, in addition to observations and comments which have explanatory value and provide some measure of perceived usefulness.

6.1 Content consolidation, periodic assessment and workshops

Replacing the single end-of-course submission with three smaller staged submissions

Despite the decision to reduce the number of assessed exercises in 2014/15, learners still struggled to complete the assigned exercises each week. These exercises were intended to contribute to the submission of one collated portfolio at the end of the course. This failing would then lead to a growing backlog of uncompleted exercises, resulting in an unmanageable workload concentrated into a short block of time towards the end of the course. Also noteworthy in 2014/15 was the high number of student non-submissions (NS) recorded. In response to these observations, a decision was made to change the assessment arrangements for the 2015/16 academic year. The end of course submission (defined as CW1) was newly divided into three separate periodic submissions (Part A, Part B and Part C), in an attempt to make the assessment work more manageable for students, encouraging more timely completion of exercises and hopefully reducing the NS rate. Each of these three periodic submission points typically required that three or four weeks of independent exercises or project tasks were completed.

Introduction of workshops

A workshop session was introduced into the curriculum in each week immediately preceding one of the three submission deadlines. No additional teaching content was delivered during these three workshop sessions; however, students could ask tutors for guidance. These sessions were supervised by a tutor and provided a focused environment for students to complete outstanding exercises before submitting them the following week.

Consolidation of teaching material

Creating three workshop weeks also inspired the rearranging of teaching content to align with periodic submissions. This change was informed by student feedback as they wanted to progress through the basic concepts faster. In 2014/15, the coverage of some concepts was drawn out over two teaching weeks. Therefore, in 2015/16, the content of W1 and W2 was merged into one week (W1), as was the content of W3 and W5 (which became W2) (Table 6.1) Thus, coverage time for the three basic programming constructs was reduced from five weeks to three dedicated weeks.

Table 6.1 illustrates the reorganised curriculum featuring workshop sessions and periodic assessment points after three to four weeks of teaching.

Table 6.1. The Semester 1 programming course curriculum before and after the 2015/16 modifications.

Course content of 2013/14 and 2014/15 versions of the course	Week	Course content from 2015/16 onwards
Variables and assignment	W1	Variables, assignment, input & output
Input and output	W2	Iteration (for, while, do while loop)
Iteration part 1 (for and while loops)	W3	Selection (if, else, else if, switch, AND/OR)
Selection	W4	Assignment workshop for Part A
Iteration part 2 (do while loops)	W5	Functions (CW1 Part A submission)
Functions	W6	Parameter Passing
Parameter Passing	W7	Arrays
Reading week	W8	Project
Arrays	W9	Assignment workshop for Part B
Project	W10	C# part 1: recap on variables, assignment, input & output (CW1 Part B submission)
C# part 1: recap on variables	W11	C# part 2: recap on sequence, iteration and selection
C# part 2: design process	W12	C# part 3: Object Oriented Principles
Christmas Break		Christmas Break
C# part 3: recap on Sequence, Iteration and Selection	W13	Assignment workshop for Part C
C# part 4: Object Oriented Principles	W14	CW1 Part C submission
CW1 submission	W15	

The impact of course delivery changes on attendance

The strong association between attendance and L4 programming grade found in Section 5.2 provided a rationale to develop interventions with the aim of increasing participation in the timetabled taught sessions. Figure 6.1 illustrates a moderate relative improvement in attendance for the post-2015/16 student cohorts in the middle section of the introductory programming course. Even though the more recent groups are not immune from declining attendance as the course progresses (many students continue to return home in the week before the Christmas break), attendance does appear to be more consistent for the most part. Whilst it is not possible to confirm whether these changes had a direct cause-and-effect influence on attendance, the insertion of an early summative assessment (CW1 Part A expected in Week 5), may have encouraged students to stay engaged at this point in the course (compared with the 2013/14 and 2014/15 years). Also, in Week 5 students are issued with a new printed study pack (exercises relating to CW1 Part B), providing some extra incentive to attend this session.

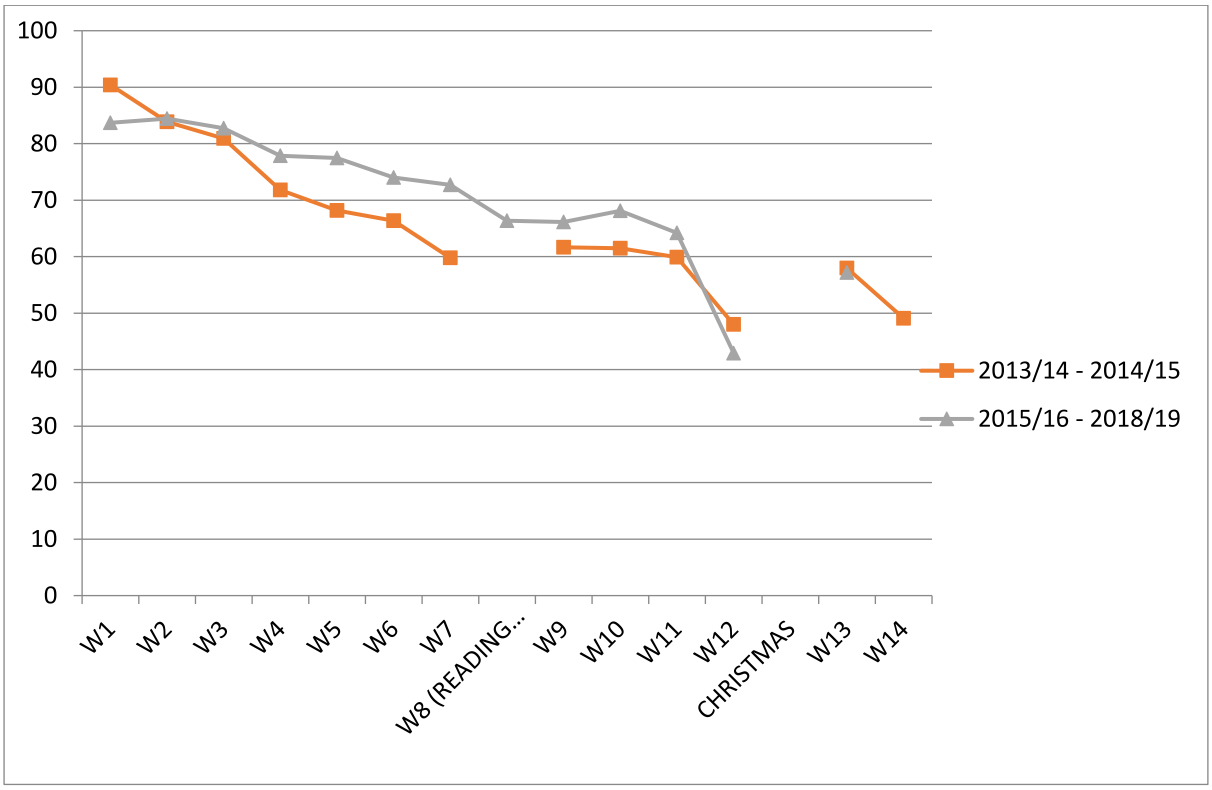


Figure 6.1. Attendance for L4 programming cohorts (pre and post 2015/16 changes).
Notes: (1) Students with abnormally low attendance (less than 40%) were regarded to be exceptionally disengaged outliers and were excluded from the analysis; (2) academic years prior to 2015/16 feature a reading week in W8 but also have an additional session in W14, whereas post 2015/16 years do not feature a reading week in W8 or a taught W14 session.

The impact of course delivery changes on non-submission rate

As previously stated, approximately one third of students failed or dropped out of the course in 2013/14 and 2014/15 (Bucks 2014a; Bucks 2014b; Bucks 2015a; Bucks 2015b). Table 6.2 illustrates considerable variation between recent academic years in the proportion of non-submissions (NS), and those who failed (achieved below 40% in course assessment). Whilst Table 6.2 does indicate a general reduction in the non-submission rate for post-2015/16 cohorts, investigation found that this is likely to be the result of more accurate monitoring of student status rather than being a direct consequence of course modifications. Prior to the 2015/16 academic year, all enrolled students who did not submit were listed as an NS, but this did not distinguish between those who enrolled and never attended, those who withdrew or those who were listed as retaking the course. However, participation and enrolment status are monitored more effectively for post-2015/16 cohorts; therefore, learners who withdrew, deferred or retook the year (but did not submit) are no longer listed as NS and no longer distort the NS rate.

Table 6.2. Submission and failure rates for L4 introductory programming course 2012-18.

Academic year	Number of students recorded on the final board	Number of submissions (A-F)	Number of non-submissions	Failure rate (%) (graded below 40%)
2012/13	58	57	1 NS (0 unlisted)	25%
2013/14	53	48	5 NS (0 unlisted)	14.89%
2014/15	92	65	27 NS (0 unlisted)	15.87%
2015/16	119	84	9 NS (26 unlisted)	15.47%
2016/17	112	80	8 NS (24 unlisted)	15%
2017/18	88	64	13 NS (0 unlisted)	11.62%
2018/19	49	47	2 NS (0 unlisted)	22%

Note: More accurate monitoring of student status was introduced for the 2015/16 academic year. This meant that those who withdrew were no longer counted as 'NS'. Instead they were counted as 'unlisted'.

Nevertheless, Table 6.3 demonstrates that the failure and non-submission rates for 2015-2018 academic years (end of course submission) are less than that for the 2012-2015 years (periodic submission).

Table 6.3. Non-submission rate and failure rate before and after periodic assessment changes.

Academic years	Number of students	Non-submission rate (%)	Failure rate (%) (graded below 40%)
2012-2015	204	13.50%	19%
2015-2018	367	8.39%	16.02%

The impact of summative assessment changes on failure rate

Further examination of periodic assessment grades (Parts A, B and C) reveals that of those who did fail the course overall, 75% passed one of three assessment parts (Table 6.4). Approximately 10% of those who attempted all three assignment parts failed on their aggregate grade.

Table 6.4. Students who failed the L4 semester 1 programming course since 2015/16 changes.

Parts attempted	Percentage of students who failed
Attempted Parts A, B and C but failed overall	10.34%
Attempted Part A only	20.69%
Attempted Part A and B, but not Part C	44.82%
Passed at least one of the Parts (A, B or C)	75.86%

Note: n for students who failed between 2015-2018 is 29.

Investigation of the trends pertaining to the assessment parts of failing students revealed that some (approximately 20% of students) submitted Part A but not Part B or Part C. Most of these students passed Part A (only one failed) which indicates that they have potential to pass the remaining Parts B and C. However, conversations and email correspondence revealed that, for some, personal circumstances prevented them from completing and submitting the work.

Nearly half of failing students (45%) attempted Parts A and B but not Part C. Whilst some narrowly passed having submitted Part A and B, there is a common mindset that it is sufficient only to achieve the minimum 40% pass requirement during the first year (Tomlinson 2014) and this tends to be characteristic of the surface learning strategy. Indeed, tutors of other L4 courses at Bucks, with multiple periodic assessments, have observed a similar trend (Luker 2019). The findings of Section 5.2 suggest that the same students score consistently across all subjects.

Impact of course delivery changes on average programming grade

Even though there is no clear signal to the effect that the periodic submission has helped those students most vulnerable to failing or has directly addressed the NS rate, Table 6.5 demonstrates that average grades did experience an improvement for the period between 2015/16 and 2017/18, when class exercises (some 60 exercises) were not assessed alongside the required independent exercises, in comparison to the earlier period.

Table 6.5. Comparison of average grade before and after periodic assessment.

Academic years	Average grade
2012-2015	56.24%
2015-2018	62.53%

Note: n for 2012-2015 intakes is 204; n for 2016-2018 years is 367.

Figure 6.2 illustrates moderate reductions in the numbers failing (achieving both E and F grades), and for those only narrowly passing with a D grade. Whilst B and C grade frequencies remain similar, there is a moderate increase in the numbers of those achieving the top A grade. The reduction of exercises (from some 100 to 40 exercises) for assessment between 2014/15 and 2017/18 may have enabled more motivated, focused and experienced students to achieve the top grade.

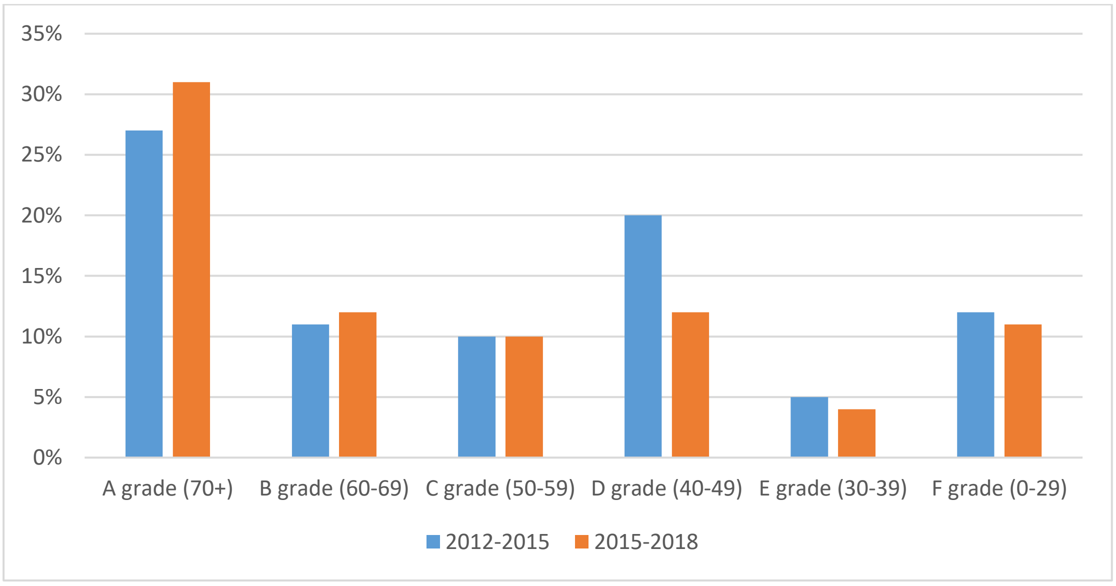


Figure 6.2. Grade distribution before and after periodic submission.
Note: n for 2012-2015 intakes is 204; n for 2016-2018 years is 367.

<u>Concerns of over-assessment</u>

An internal review suggested that having three separate assignments was perhaps too much summative assessment at L4. In most circumstances, three unique assignments would indeed create a higher workload than one assignment. However, in this case, the contents of one assignment are simply divided into three smaller parts and submitted periodically rather than together. In fact, due to the consolidation of teaching materials, the total number of exercises spread across three assignments is less than the number previously required for the one assignment submitted at the end of the semester.

Lost feedback opportunities

Whilst all tutors instructed their students to submit their work in intervals, some tutors did not grade the three assignments until the end of the semester (per the previous assessment format). Failure to monitor submission rate negates the opportunity to identify vulnerable students early and intervene accordingly, as well as the opportunity to provide feedback for student development throughout the course. The strength of measurable signals for the effectiveness of this intervention is thus diluted.

Tutors accustomed to the previous assessment format may have been less aware of the potential benefits of the staged assessment process, and may also have found it difficult to switch from an assessment regime that had been largely unchanged for the past decade. Convincing tutors to adopt new practices is challenging (Barker et al. 2015; Guzdial 2016). However, communication with the teaching team revealed that the lack of time during term time was a significant obstacle to providing significant levels of constructive feedback to students. Further work is needed to ensure that course instructors have sufficient time and resources for the effective and timely delivery of feedback.

Development of summative assessment

Whilst the development of a summative assessment mode which more accurately monitors understanding and differentiates between learning approaches (Section 5.1 & 5.3) is important, this may not address the underlying motives that pervade student engagement with their course. Indeed, all conventional assessment modes (written or oral examinations, small-scale lab exercises and projects) are vulnerable to surface level memorisation techniques.

Biggs and Tang (2011: 6) state that problem-based learning activities (which are examples of active learning) have greater potential to transition learners from lower-order thinking - associated with performance motives and surface learning strategies - to higher-order thinking (commonly associated with mastery motives and a deep learning approach). However, this L4 programming course already features over 100 small-scale exercises of a type and difficulty thought to be more appropriate for novice programmers, who exert lower-order thinking skills (Teague 2015: 265).

Therefore, further questions arise regarding the alignment of these small-scale lab exercises to better facilitate change in motives (Biggs & Tang 2011) and change of approach (Adey & Shayer 1994) towards a deeper understanding of programming concepts. This is addressed further in Section 6.3.

However, the next section (Section 6.2) discusses the role of unassessed formative quizzes for monitoring understanding of programming concepts and their usefulness for predicting L4 grades.

6.2 The introduction of unassessed quizzes

Analysis of Week 4 and Week 9 quiz questions for the 2015/16 academic cohort

Further to the three course curriculum changes discussed in Section 6.1, and in response to student requests (Section 5.1), two unassessed formative quizzes (see Appendix 23 and Appendix 24) were introduced in the 2015/16 academic year. The Week 4 (W4) quiz included 20 questions on the programming concepts introduced in the first three weeks (sequence, selection and iteration) and the Week 9 (W9) quiz contained ten questions on the material covered in weeks 5-7 (functions, parameter passing and arrays) and ten questions from weeks 1-3. The W9 quiz featured five identical questions from the W4 quiz. Table 6.6 illustrates that the percentage of correct answers rose between W4 and W9 for four out of the five repeated questions, the bottom listed question being the exception (Examples of iterative statements).

Table 6.6. A comparison of correctly answered questions in W4 and W9.

Question description	Correctly answered in W4	Correctly answered in W9
"What are the three main programming constructs?"	96%	100%
"Declaring a variable and assigning a value is known as…"	48%	70%
"What does concatenation mean?"	42%	65%
Variable definitions require a data type and an identifier	65%	70%
Examples of iterative statements	48%	35%

Note: n for W4 completed quizzes is 42, and n for W9 completed quizzes is 33.

Table 6.6 demonstrates that students experienced most difficulty when required to differentiate between selection statements ('if' and 'switch') and iterative statements ('for', 'while' and 'do while'). Further analysis revealed that, whilst nearly 70% of students chose one of the correct options in W4, few of those 70% chose all three correct answers. In the W9 quiz, 38% of respondents thought that the 'if' statement was an iterative statement, and 15% thought 'switch' was an iterative statement. Tutors have frequently overheard students referring to 'if loops' during practical sessions. This further illustrates that this an area of confusion and misconception.

Pearson's correlation analysis indicates that there is a moderately strong but highly significant relationship between the 2015/16 W4 quiz score and Part A grade (r = 0.549, p = 0.001, n = 42). The 2015/16 W9 quiz score was also both strongly and significantly correlated with Part B grade (r = 0.462, p = 0.007, n = 33). Pearson's correlation analysis also indicates a very strong and positive relationship between the W4 quiz score and the W9 quiz score (r = 0.785, p = 0.001, n = 32).

Performance in the W4 quiz is also a strong and significant indicator of a student's final grade performance (r = 0.466, p=0.002, n = 42).

Corney and colleagues (2011) also discovered that a W3 test was predictive of final L4 programming grade, as did Robins (2010) who found that those who excelled in initial tests tended to do well in final exams, and those who struggled initially did not tend to improve.

Analysis of the 2016/17 and 2017/18 weekly quiz scores

In 2016/17 and 2017/18, the quiz format was changed from two relatively large quizzes (each with 20 questions) to smaller, more frequent, weekly quizzes (each with 8 questions), in order to more closely monitor student understanding of topics introduced each week.

Analysis of responses to questions continued to illustrate the difficulty that students had in differentiating between selection and iterative statements. In the 2016/17 W3 quiz, 43% of students thought the 'if' statement was an iterative statement, and 30% thought the 'switch' statement was an iterative statement. In the 2017/18 W3 quiz, 25% thought 'for' was a selection statement, and 20% thought 'while' was a selection statement.

Interestingly, approximately 88% of both 2016/17 and 2017/18 students selected one correct answer in response to a question on 'do while' loops that required all correct characteristics to be chosen. However, only 45% of both academic years selected a second correct answer**.** Failure to select all three correct answers resulted in the attempt being marked as incorrect. This requirement may explain why it was the most incorrectly answered question of the quiz.

Students also struggled to correctly answer array questions in both the 2016/17 and 2017/18 W7 quizzes. Only 25% of the 2016/17 respondents correctly indicated how arrays are passed between functions. The three remaining incorrect answers were equally divided amongst the three remaining multiple-choice detractors (approximately 25% each). Of the 2017/18 year, 43% thought the array name contained memory addresses for every element. This is close, but not correct. Another 36% of 2017/18 learners incorrectly thought that the arrays were passed 'by value'.

Three weeks of quiz scores were combined and compared with the corresponding Part A and Part B grades. There is a moderate correlation between W1-3 quiz score and Part A grade for the 2016/17 year (r = 0.309, p = 0.198, n = 19) and a strong and significant relationship for the 2017/18 cohort (r = 0.754, p = 0.001, n = 22). However, there are moderate but non-significant correlations between W5-7 quiz score and Part B grade for both 2016/17 (r = 0.414, p = 0.160, n = 13) and 2017/18 (r = 0.387, p = 0.113, n = 18).

Nevertheless, cumulative quiz scores were found to be strongly and significantly correlated with final grade. Pearson's Correlation analysis reveals a strong and significant correlation between W1-3 quiz total and final grade; for 2016/17 (r = 0.655, p = 0.003, n = 18) and for 2017/18 (r = 0.555, p = 0.007, n = 22). Although there is a strong and significant relationship between W5-7 quiz total and final grade for the 2016/17 cohort (r = 0.592, p = 0.042, n = 12), there is only a moderately strong but non-significant relationship for 2017/18 (r = 0.453, p = 0.059, n = 18).

It is interesting that the W1-3 and W5-7 quiz totals correlate strongly and significantly with final grade but not with Part A and Part B grade for the 2016/17 and 2017/18 cohorts (with one exception). Strong and significant correlations between Part A grade and Part B grade for 2016/17 (r = 0.705**, p = 0.001, n = 45) and 2017/18 (r = 0.580, p = 0.004, n = 23) indicate that students scored consistently across their assignments. However, there is a difference between the periodic format adopted in 2015/16 and weekly quiz format of 2016/17 and 2017/18. Periodic measurement of programming knowledge in W4 and W9 (in 2015/16) captures development in understanding over the period of three to four weeks, whereas totalling weekly quiz scores does not capture progressive understanding after the session in which the topics were introduced.

It would be desirable to issue both weekly quizzes (W1-3; and W5-7) to progressively monitor the understanding of each topic, in addition to larger periodic quizzes (W4 and W9) to evaluate the degree to which this knowledge is retained or improved upon.

Longitudinal comparison of weekly quiz scores

Figure 6.3 illustrates that the average quiz score for W2 (iteration) and W7 (arrays) was approximately 10% lower than for other weeks. Students have historically struggled to understand arrays due to multiple ideas and syntax being introduced; such as the subscript operator, pointers to memory addresses and referring to elements via an index (Teague 2015: 266). Furthermore, exercises typically require students to apply array concepts in combination with iterative statements for efficient array content input and retrieval, and to pass arrays to and from functions.

This may also be the case with iterative statements. These not only require variables and assignment to be understood from W1 (and to be applied with iterative statements), but also introduce new ideas such as evaluating conditional statements (to true or false) and repetition of specific statements (within braces). In addition, the 'for' loop operates uniquely to other iterative statements – with exceptions to syntax rules (no semi-colon after the '++' in C based and Java languages) and the order in which statements are executed.

It is possible that lower than average scores in W2 and W7 may indicate that the topics being introduced require too many cognitive resources to be learnt at the same time (Sweller 2011). New topics also often depend upon and assume that concepts introduced in previous weeks have been fully understood and assimilated – which is often not the case (Teague 2015). Therefore, Section 6.3 also goes on to discuss whether teaching concepts in isolation from previous weeks (Luxton-Reily et al. 2017), in addition to further decomposition of the components of iterative statements and arrays (Teague 2015: 266) would enable more effective learning.

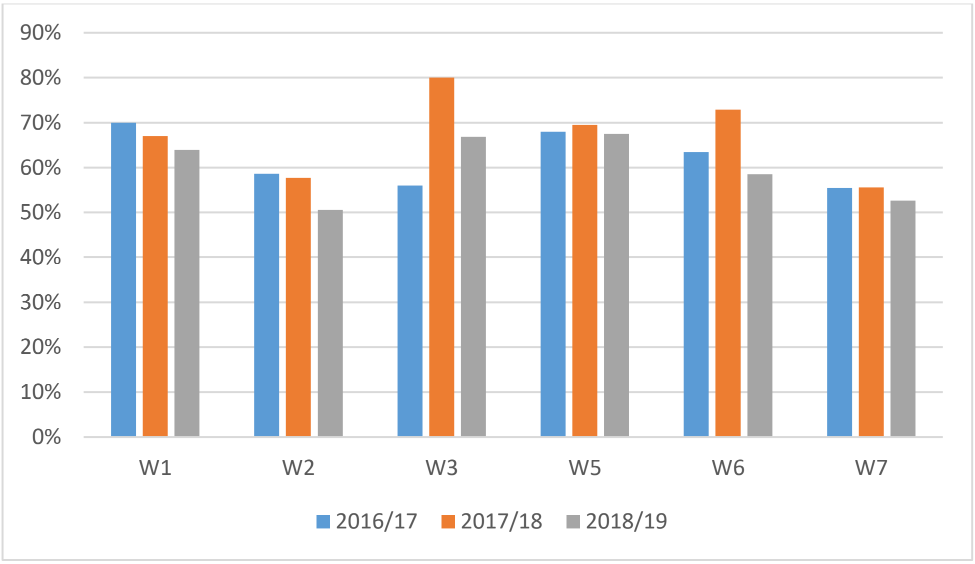


Figure 6.3. Average weekly quiz scores for the 2016/17 to 2018/19 cohorts.
Notes: (1) number of questions in 2016/7 quiz is 4 and 2017/18 and 2018/19 quiz is 8; (2) number of completed quizzes 2016/17 is 54; n for 2017/18 is 43; n for 2018/19 is 38.

<u>Limitations of multiple-choice questions</u>

Multiple choice questions are appealing to educators for the fact that they can be graded automatically. However, similarly to summative assessment, multiple choice may not capture the wealth of approaches adopted to answer questions and solve problems. Multiple choice tests are susceptible to guessing (one student openly admitted that he had guessed most of the questions on one quiz) or plagiarism (typical of the collaborative environment investigated in Section 5.3; students have been observed to ask their peers for the correct answer to the quiz). La Barge (2007) recommends the inclusion of an additional 'tickbox' for each question so participants can indicate whether they knew or guessed the answer. This would prevent distortion of results. Other techniques such as randomising the order of questions and answers, in addition to instructor intervention, can help reduce the temptation for peers to copy from each other.

Differences in participation between groups and tutors

Whilst the tutors at Bucks strive for consistency by teaching from the same slides, assigning the same exercises to be assessed and using the same marking criteria, there are subtle differences between teaching style and the dynamics of different groups.

The researcher actively encouraged his students to take the quizzes each week (resulting in a moderately high participation rate). However, levels of participation varied between groups taught by other tutors and sometimes declined over time. The study was unable to determine whether this was due to differences in the approaches used by tutors to encourage greater engagement or due to differences between the motivation levels of student groups. Students motivated by performance goals and surface learning approaches would be less inclined to engage with optional and unassessed activities.

Effectiveness of formative assessment for addressing underlying motivations

Unassessed formative assessment can help learners manage their learning process without the distraction of letter or numerical grades. However, this assumes that learners are primarily motivated by mastery and want to engage in unassessed learning opportunities. As previously discovered, many wish to engage with their studies, but those only concerned with passing the course with minimal effort may not choose to engage. Whilst Oxford University is renowned for its tutorial system, which is said to issue ten formative assignments for every one summative assignment (Gibbs 2015), the success of this system is attributable to high levels of student motivation.

Despite revealing key student behaviours and patterns of engagement, the findings of Section 6.1 and 6.2 suggest that formative assessment and periodic summative assessment have not addressed the issues they were specifically introduced to target. An appropriate balance between formative and summative assessment is likely to be characteristic of an aligned curriculum which inspires deep learning (Biggs & Tang 2011; Matheison 2015). However, further intervention is required to more effectively aid the transition in motivation and mindset - from performance goals associated with the surface learning approach to mastery goals associated with deep learning approach. This discussion is continued in Section 6.3 which considers the format of the teaching session and the influence of the tutor.

6.3 Towards a revised ‘incremental’ teaching model

The conventional format of teaching sessions (one-hour lecture followed by two-hour practical) and existing content was re-arranged (see Figure 6.4) in an attempt to increase engagement and better enable the development of programming ability amongst students. These changes are explained below and are underpinned by learning and psychological theories.

Shorter teaching segments

Firstly, the material typically communicated in a one-hour lecture was divided into separate shorter increments of teaching, corresponding to parts of a programming concept. Cognitive strain is reduced by only needing to concentrate on one or two new ideas at one time (Sweller 2011; Vygotsky 1978). Successful understanding of a teaching segment provides the foundation for the next teaching segment and facilitates the incremental but manageable construction of mental models of concepts. Shorter teaching sessions are also favoured as students are rarely able to sustain their attention for an hour. Studies demonstrate that the average attention span is limited to between three and fifteen minutes (Davidson 2011; Rosen et al. 2011: 124). Although attention span can be extended with training, disengagement with lecture materials will adversely affect knowledge retention (as reported in Section 2.2.4). Therefore, this model attempts to regulate presentation of knowledge to shorter intervals for increased attentiveness.

Pairing shorter teaching increments with exercises

Each shorter teaching session is immediately followed by a small exercise corresponding to the concepts communicated. This is one of the same exercises which would have been undertaken during the practical sessions that follows one-hour lectures. Each learner in the class has the opportunity to actively put the idea they have just heard into practice (Biggs & Tang 2011: 6; Freeman et al. 2014). The change in activity (from listening to coding; passive listening to active learning) provides new stimulus and thus helps to ‘refocus’ learners’ attention (Gazzaley & Rosen 2016: 228). Applying the content in an active way also repeats/reinforces the ideas (Davidson 2011). Tutors also play an active part by providing individual help for students during this time. Correcting misconceptions is essential before attempting the delivery of new content that attempts to extend upon the previous ideas. Pairing shorter teaching sessions with appropriate exercises (constructive alignment) demonstrates the relevance of exercises and the essential part that they play in learning (Biggs & Tang 2011; Matheison 2015). Successful completion of small exercises early on also develops confidence, momentum, and self-belief (Robins 2010), all of which increase motivation.

Demonstrating the problem-solving process

After allowing appropriate time for students to attempt activities by themselves, under the supervision of a tutor, the tutor can then deliver a fully interactive walkthrough of solutions. The instructor explains how they would apply their strategy to solving the same problem recently attempted by students - typically via three stages: (1) data input (variables and assignment); (2) modification or application of data (calculations, passing to functions etc.); and (3) data output (printing results to screen) - but involving students by asking questions such as *'what data do we need to store?'* and *'what data do we need to output?'* and inviting additional questions and discussing steps of the problem-solving process. It is also an opportunity to revisit and reinforce ideas communicated in earlier teaching segments. The introduction of demonstration activities was inspired by successive studies (Bennedsen & Caspersen 2005; Guzdial 2016: 11) recommending that tutors explicitly teach their problem-solving process to guide learners who do not know how to start and approach the development of their own solutions.

These three steps (presenting new information, an exercise, and demonstration of the solution) are repeated for introducing parts of a programming concept sequentially. An example of this is provided below in Figure 6.2. The example also demonstrates the use of an interactive recap session (Q&A and discussion) for the purposes of reminding students of previous content before beginning the next topic. This is informed by constructivist philosophy; which emphasises connections between topics (Biggs & Tang 2011: 67). Sessions also include quizzes before the teaching, activity and demonstration cycle begins and again at the end of the session. The quiz questions are identical but only the end of session quiz provides feedback on correct and incorrect answers. As well as providing formative opportunities for students, tutors may also use quizzes to monitor learning progress.

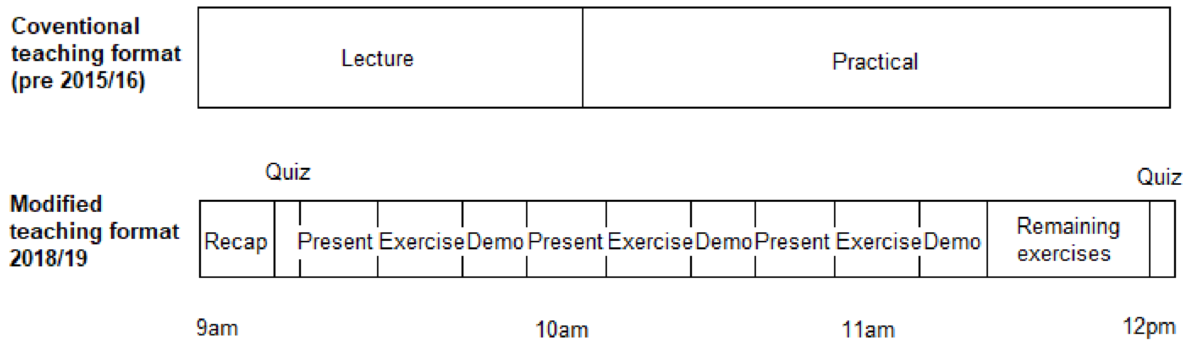
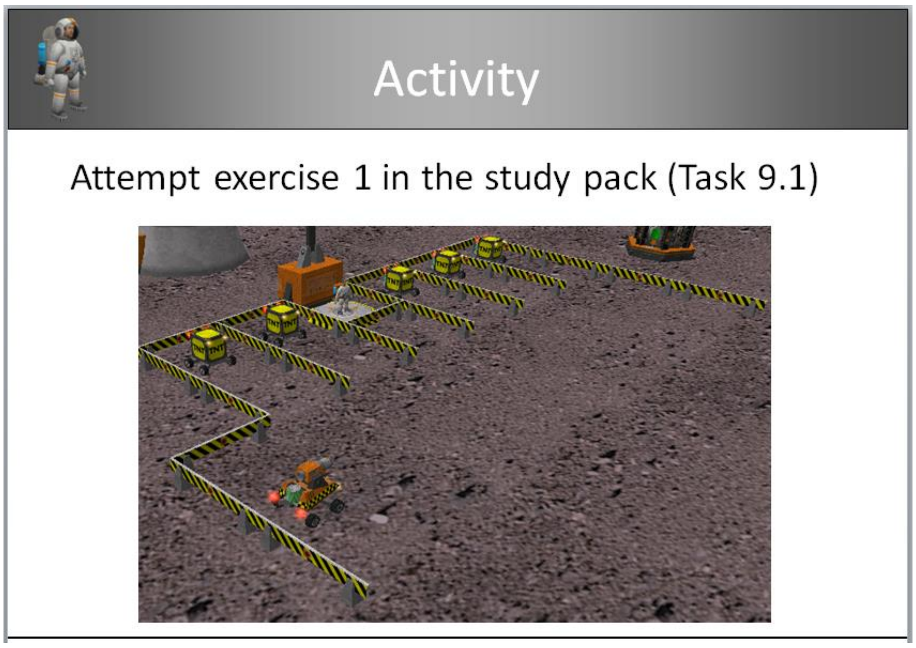


Figure 6.4. Comparison of the conventional and new teaching model.

An example of the incremental teaching arrangement

This example describes how the concept of selection is taught using the cycle of three short activities (teaching, activity and demonstration). The first increment introduces the ‘if’ statement in addition to the comparison operator and conveys example scenarios using a mixture of data types (comparison of integers, strings, and Boolean data) to demonstrate the different code paths for true and false conditions. Students are then instructed to work on their first exercise which requires them to repetitively control a robot to move, turn and shoot five targets placed in a simulated 3D landscape (Figure 6.5). However, they have to use one ‘if’ statement to avoid shooting an astronaut in the middle of the sequence.



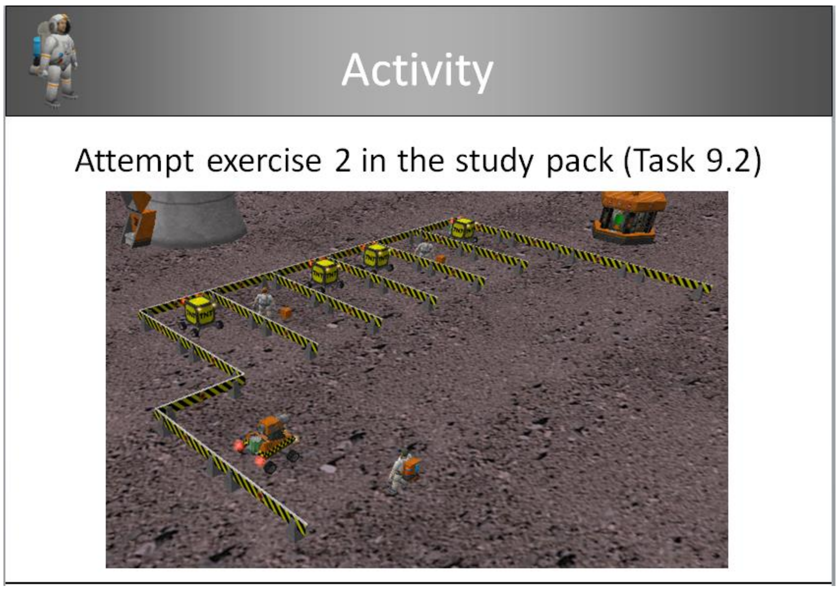
Activity

Attempt exercise 1 in the study pack (Task 9.1)

Figure 6.5. Slide 16 from W3 presentation, instructing students to attempt the first exercise in the study pack.

Following an attempt at the first exercise and interactive demonstration, the next teaching cycle extends the basic ‘true and false’ idea of the if statement to combine two conditions using logical OR and AND operators, including coded examples demonstrating the differences between the two operators. The second exercise requires students to shoot four targets, this time avoiding shooting at two astronauts (see Figure 6.6). Tutors observe that students commonly and incorrectly use the AND operator to avoid the two astronauts (do not fire at position two and five). However, because the robot cannot be at position two and five simultaneously, the OR operator is required. This exercise is therefore effective at both, demonstrating the difference between the two operators discussed in the previous teaching block, and encouraging further discussion of code errors to promote deeper understanding of concepts.

Analysis of formative quizzes indicated that discussing mistakes did indeed enhance understanding as most students correctly answered the difference between AND and OR operators in the post-teaching quiz.

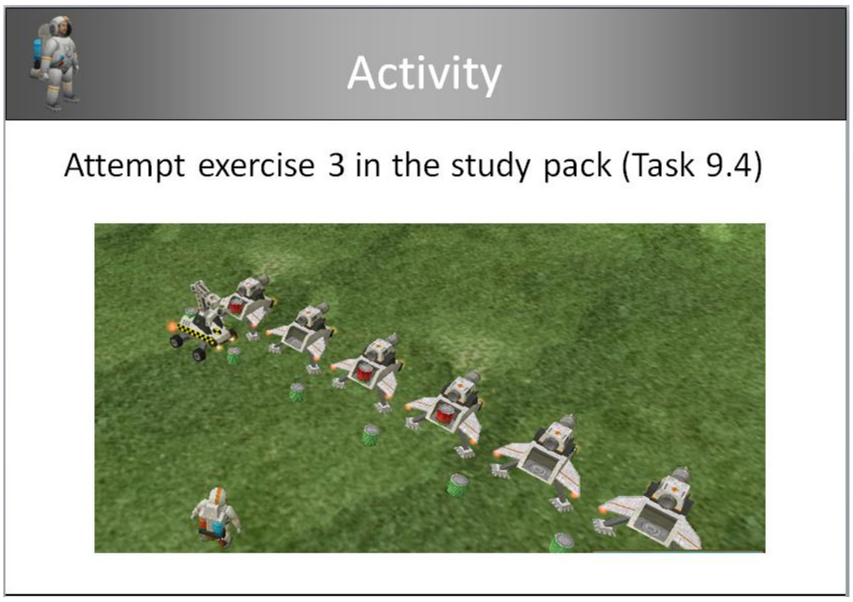


Activity

Attempt exercise 2 in the study pack (Task 9.2)

Figure 6.6. Slide 21 from W3 presentation, instructing students to attempt the second exercise in the study pack.

In the final presentation step of the session, the 'else' statement and the 'else if' statement are explained in the context of a block of selection statements. Students are given one last prescribed exercise (see Figure 6.7) for them to implement 'else' and 'else if' statements. This is then followed by a complete demonstration of the exercise.



Activity

Attempt exercise 3 in the study pack (Task 9.4)

Figure 6.7. Slide 28 from W3 presentation, instructing students to attempt the third exercise in the study pack.

Quantitative analysis of quiz results pre- and post- teaching

Table 6.7 demonstrates that the post-teaching quiz scores are greater than the pre-teaching quiz scores by an average of at least one point for most weeks. The general decline in gain, over several weeks, is expected due to the increasing complexity of concepts and their combination. Despite differences in standard deviation values between different weekly quizzes, the difference between pre- and post- teaching quizzes is minimal. This indicates that the spread/dispersion of grades remains consistent and, although not tested, also suggests that students make incremental improvements of similar magnitudes.

Table 6.7. Pre- and post- teaching quiz averages for the 2018/19 students.

Week	Pre-teaching quiz average	Pre-teaching quiz Std. Dev	Post-teaching quiz average	Post-teaching quiz Std. Dev	Average increase
W1 (Sequence)	4.6	2.021	5.9	2.006	+1.3
W2 (Iteration)	3.45	1.601	4.9	1.833	+1.45
W3 (Selection)	5.7	2.136	6.4	2.065	+0.7
W5 (Functions)	5.5	1.878	6.4	1.512	+0.9
W6 (Parameters)	5.55	1.751	5.778	1.986	+0.278
W7 (Arrays)	4.22	1.650	5	1.333	+0.78

Notes: (1) number of questions in each quiz is 8; (2) number of participants is 14.

Further investigation revealed that those without prior programming experience gained more in the initial weeks covering basic concepts (W1-3) than those with more experience (see Figure 6.8). However, those without prior experience did not benefit from the same magnitude of progress in later weeks (W5-7). On the other hand, the most experienced programmers benefited least in the initial weeks (although they did maintain consistently higher scores) but gained more in later weeks.

Cognitive theory (Sweller 2011) may help explain this trend, as those with experience may be able to apply the basic concepts with minimal cognition or automatically. This allows them to use their cognitive resources to understand less-familiar topics introduced later in the course. However, novices do not have this advantage and their cognitive resources quickly become challenged and overloaded.

Furthermore, Figure 6.8 also illustrates a reduction in W6 and W7 post-teaching quiz scores (compared with pre-teaching quiz scores) for the least, and moderately experienced learners. This decline may indicate inconsistent understanding (mental models) of those concepts taught later in the course.

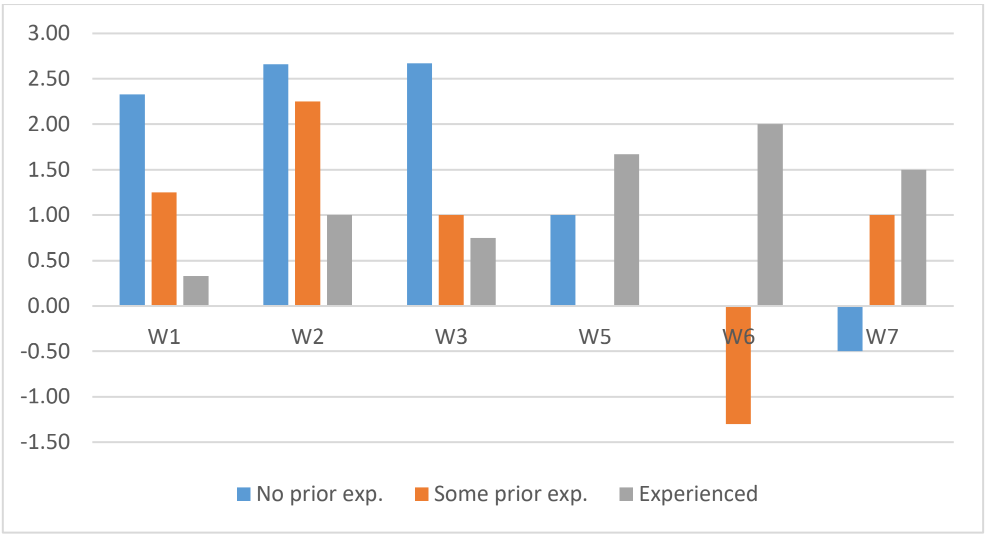


Figure 6.8. Learning progress of 2018/19 grouped by level of experience.
Notes: (1) n for no prior experience group is 4, n for moderate experience group is 4, and n for the most experienced group is 5; (2) Level of prior experience calculated from survey at the start of the course (Appendix 13); and W14 code knowledge captured from survey (Appendix 1) at the end of the course.

The 2018/19 students completed the end of course survey (identical to the survey referenced in Section 5.1 - available in Appendix 1). Attempts at the four questions assessing code knowledge (for the basic concepts of sequence, selection and iteration) were taken as a measure of how well students retained this knowledge at the end of the course (W14). Figure 6.9 illustrates that all three categories of student (least, some and most prior programming experience) managed to retain knowledge imparted during the initial weeks of course delivery.

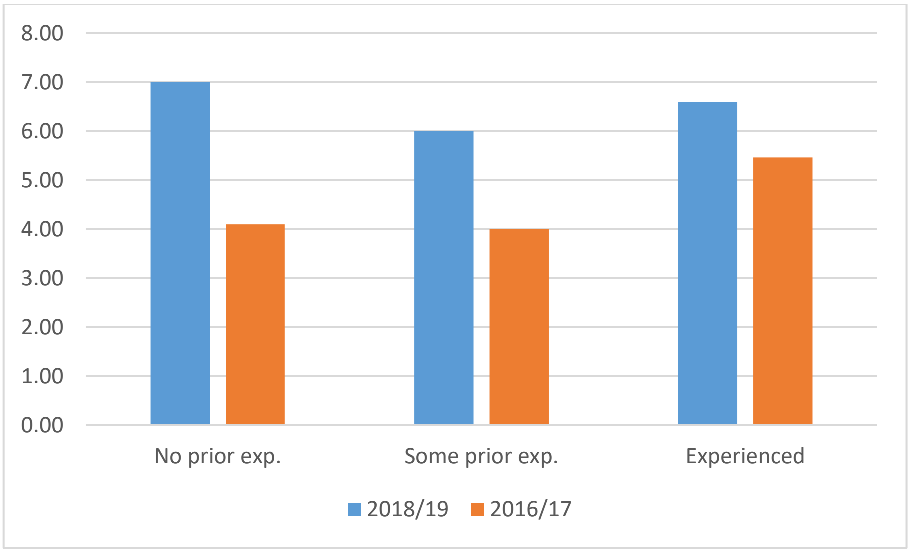


Figure 6.9. A comparison of W14 code knowledge test scores.
Notes: (1) W14 code knowledge score is out of 8; (2) n for no exp. 2016/17 is 21; n for some exp. 2016/17 is 7, n for most exp. 2016/17 is 13.

Students subjected to the new incremental teaching interventions also achieved higher average marks in the end of course knowledge test than those for all earlier academic years (Table 6.8).

Table 6.8. Average code knowledge scores by academic year.

Academic Year	Average code knowledge score
2014/15	4.36
2015/16	4.51
2016/17	4.57
2017/18	N/A
2018/19	6.3

Notes: (1) data for 2017/18 was not collected; (2) scores calculated from survey are presented in Appendix 1.

<u>Student behaviours and responses to the introduction of incremental teaching</u>

The researcher observed greater engagement with the presentation, practical exercise, and demonstration steps of teaching increments. Before the introduction of this approach, students were commonly observed to leave early or disengage with the lecture and practical, and were easily distracted to browse websites, look at their phones and talk about unrelated subjects. However, shorter activities appear to have helped students refocus and stay engaged with learning material. This is consistent with theories regarding attention span (Davidson 2011) and that greater engagement, although not necessarily indicating deeper understanding of programming concepts, is required to enable learning (Biggs & Tang 2011).

Student comments suggest their acceptance of the new model for incremental delivery as being effective in engaging them in their studies and for encouraging attendance. A conversation transcribed from observation notes during a practical session (see Fragment 6.1 below) describes the difference between teaching techniques and their perceived usefulness.

Fragment 6.1:

> *S11: "…I learn so much more through your teaching methods. Some tutors just read bullet points off the screen and I don't learn from that."*
> *{S12 nods in agreement}*

Biggs and Tang (2011: 18) refer to the reading of bullet points as being characteristic of, what they term, 'Level 1 teaching', in which the educator sees his or her primary responsibility to transmit knowledge. However, 'Level 3 teaching' is more dynamic in communicating content that is relevant to the current level of a learner's understanding. This is consistent with the ZPD theory (Vygotsky 1976) of extending on a foundation of existing knowledge and referenced in Section 5.3.

Additional comments made by students in practical sessions (Fragment 6.2 and Fragment 6.3) also illustrate the perceived value of demonstrations and the communication of process. Fragment 6.3 provides evidence that demonstration and active learning can influence change in learner motivation and approach; corroborating Biggs and Tang's (2011: 6) work.

Fragment 6.2:

> *S13: "I find your problem-solving demonstrations really helpful – especially how you let me ask questions at any point in the demonstration. I learn so much in your classes"*

Fragment 6.3:

> *S14: "Through demonstrating alternative ways to solve the problem, you helped me to see that there is more than one way to solve a problem. You helped me to see that there is more to it than just getting the right answer…"*

<u>Limitations of methods and quantitative indicators</u>

As previously established, numerical measurements of learning success or vulnerability are, at best, imperfect indicators of progress or engagement that provide little information on causality. Thus, variations in non-submission frequencies were believed to be due largely to administrational changes (see Section 6.1) rather than changing levels of student engagement. Similarly, an increase in average grade does not necessarily indicate greater learning or improved tutor delivery.

Numerical grade measurements are also vulnerable to surface strategies such as guessing (which distort measurements of learning). Therefore, despite the positive impact of the incremental model on engagement and attentiveness, peer related behaviours that inspired greater confidence may not have been reflected by quantitative indicators.

However, the insensitivity of indicators to such behaviours is to some extent compensated by the use of mixed methods and qualitative approaches (e.g. see Section 5.3). This does not detract from the value of the incremental model or the validity of quantitative analyses, but recognises that qualitative approaches informed by social learning theory are also important for deeper understanding of possible causal mechanisms behind the indicators used. Thus, the contribution of peer-learning (Section 5.3) within the incremental model, to creating engaging learning environments, should be celebrated and its uptake encouraged.

Although many observations and student comments were recorded from a number of student cohorts, a larger sample of completed quiz data, pre- and post- teaching, from more cohorts would have strengthened conclusions. Despite the availability of equivalent post-teaching quiz scores for earlier years (Section 5.2), the level of knowledge prior to teaching was not captured and therefore learning progress could not be gauged. Comparison of learning progress before and after introducing the incremental teaching model would have been useful for evaluating its effectiveness. Nevertheless, pre- and post- teaching quiz scores revealed trends in the learning improvements made by novice and more experienced programmers. Overall, end-of-course code knowledge test grades were greater after introducing the incremental teaching model.

Limitations of a linear teaching model

Whilst this teaching model is underpinned by constructivism – incrementally and sequentially teaching concepts – observations and comments affirm that learners desire to progress through material according to their level of experience. Those less experienced at programming required more attempts at coding, compiling and executing programs before being able to successfully complete class exercises. Attending to the needs of less experienced students sometimes also delayed the demonstration step and next teaching cycle for the remainder of the group.

To avoid delay and disadvantage to more experienced programmers, tutors could set time limits on practical exercises and then use the demonstrations to help less confident students to catch up. Additionally, tutors may encourage more advanced learners to support less experienced peers. This would help towards balancing the pace of completion and also encourage the development of student relationships towards a peer teaching culture, similar to the team dynamic that has developed naturally among Games students (see Section 5.3).

Whilst smaller groups (not exceeding twenty individuals) do allow greater access to a tutor, it is not feasible to simultaneously manage the individual development of every student in the class. Therefore, a recommendation is made in Chapter 7 to investigate adaptive content delivery and assessment systems for meeting the individual needs of students.

6.4 Chapter 6 Conclusion

This chapter addresses Objective 5 "*Suggest early forms of diagnosis and pedagogic intervention useful for retaining and motivating students whose behaviours indicate vulnerability to failing or to not progressing with their studies*" as stated in Section 1.5.

Section 6.1 outlined the introduction of periodic assessment and consolidation of teaching material to create opportunities for workshops to reduce the rate of non-submission (NS). These changes did not significantly impact the NS rate. However, the high NS rate in 2014/15 was thought to be caused by administrational changes in the more accurate monitoring and recording of student status. The introduction of phased submission points coincides with a slight reduction in those scoring less than 40% and an increase in those achieving between 40% and 49% as well as those achieving over 70%. Investigation of periodic submission data revealed that nearly half of those who failed only attempted the first two assignments (Parts A and B, but not Part C). Other individuals who did not pass made consistent progress for a period of the course but could not sustain this due to personal difficulties preventing study. The change to periodic assessment in three parts may have also encouraged greater attendance between W5 and W9 which, prior to changes in 2015/16, was prone to declining attendance.

Section 6.2 analyses revealed that formative quiz scores as early as W4 were indicative of final L4 programming grades. Further analysis of quiz scores highlighted particular difficulties in understanding the W2 and W7 topics of iteration and arrays. This in turn indicated a need to ensure that earlier concepts were properly understood before students were able to progress to more advanced subjects, and to successfully apply multiple concepts to more challenging programming exercises. Whilst quizzes were useful for guiding the development of teaching materials, the vulnerability of multiple-choice quizzes to copying and guessing was also acknowledged.

Section 6.3 described the introduction of a revised and 'incremental' teaching format that presented materials, associated exercises and review demonstrations in short and topic-focused cycles. Findings suggest that these may be more effective and engaging in delivering programming concept subjects. Analysis of pre- and post- teaching quiz scores revealed patterns of learning progress which are consistent with cognitive load theory and that knowledge of basic concepts is retained to the end of the course. Other observations and student comments indicated increased attentiveness and an overall acceptance of the incremental teaching model as a useful and engaging approach.

Chapter 7 Conclusions and recommendations for further investigation

7.1 Review of progress against the Aim and the Objectives

This research project aimed to identify predictors of programming success or vulnerability toward developing effective interventions for improving learning and teaching, retention and progression in computer programming courses (Section 1.5). Five objectives were identified for fulfilling this aim. Each objective was independently addressed in the chapters and sections as illustrated in Table 7.1.

Table 7.1. Mapping the completion of objectives to thesis chapters.

Objective number	Objective description	Chapter(s) that address objective(s)
1	Review predictive trends and student behaviours already identified in literature.	Sections 1.4, 2.1 and 2.2
2	Investigate pedagogical interventions that have been implemented in other higher education establishments.	Section 2.3
3	Evaluate the usefulness of prior qualifications for preparing students to learn computer programming in a higher education establishment.	Chapter 4
4	Explore patterns of engagement with programming learning environments towards identifying potential indicators of success and vulnerability.	Chapter 5
5	Suggest early forms of diagnosis and pedagogic intervention useful for retaining and motivating students whose behaviours suggest vulnerability to failing or to not progressing with their studies.	Chapter 6

Whilst a systematic search of databases (e.g. ACM and IEEE) could have identified CSEd literature more efficiently, the vast range of influences underpinning this research (cognitive psychology, computer science education literature, national education documents, HE and workplace trends etc.) required a broader search to be conducted. The review of literature did reveal prominent trends relating to motivation and prior educational experiences (described in Section 7.2) towards addressing Objective 1. Despite the introduction of new L2 and L3 computing and computer science qualifications during the timeframe of this research, the review of the recently superseded curricula revealed underlying issues regarding inconsistent teaching quality and expertise which exist independently of syllabi.

Drawing on a wide range of sources, Section 2.3 extended the review by evaluating a range of pedagogical interventions to address Objective 2. Whilst the techniques encouraging active learning were generally regarded to be successful in the institutions where they were implemented, there is an absence of established research approach and measuring instruments with which to evaluate the effectiveness of teaching interventions in CS contexts. However, researchers have identified a need to validate interventions at multiple institutions and for findings to be compared to general education theories in order to elevate observation of phenomena to concrete evidence (Fincher & Petre 2004).

The study reported in Chapter 4 attempted to address Objective 3 through collection and analysis of student application data from multiple institutions. Data from a partner college was advantageous for validating general learning trends, such as the consistency of learning performance across related courses. However, analysis of L6 data was unable to validate the specific L4 performance patterns of entrants with A-level and BTEC qualifications awarded in the UK. Nevertheless, the sample size of A-level and BTEC students is sufficient to qualify the patterns reported here. Whilst Pearson's correlation analysis did establish trends between student attributes and their L4 performance, on reflection, Spearman's rank-order analysis would perhaps have been more robust in circumstances where assumption of normality may not have applied.

The literature review recommended the collection of data through video and audio recordings to address Objective 4. Whilst the findings reported in Section 5.3 have contextual limitations (specific materials and instructions relating to the Ceebot IDE), the underlying principles guiding motivation and problem-solving approaches exist independently of task requirements and make a valuable contribution toward understanding patterns of undergraduate programming behaviour.

Despite the need for validation to address Objective 5, resources were only sufficient to deliver and evaluate course changes at one institution. However, the comparison between the performance of academic cohorts pre- and post- intervention reduced the influence of extraneous factors (tutor, time of teaching session, prior programming experience etc.) on results. Furthermore, efforts to ensure consistency for assessment, and to ensure that no student was unfairly disadvantaged, meant that it was not appropriate to conduct an experiment where two groups of students in one academic year were subjected to different assessment regimes. The 'incremental' teaching model was applied to multiple cohorts of small groups and the resulting observations are consistent with phenomena reported in literature (such as increased engagement with learning materials as a result of more frequent opportunities to actively participate). Although this affords a degree of confidence that similar results may be expected if this approach were applied to deliver content at other institutions, further replication would be required to confirm the effectiveness of an incremental approach elsewhere.

7.2 Contributions of this research to pedagogy and their application in the classroom

The identification of reliable indicators of L4 programming performance amongst student profile data and course engagement (Objective 1)

The review of literature established that prior programming experience was advantageous if learnt correctly. Studies that investigated the differences between A-level Computing and A-level ICT entrants tended to conclude that A-level Computing qualifications better prepared students for undergraduate study than A-level ICT qualifications. A-level Computing students often outperformed their A-level ICT counterparts in L4 programming assessments (Drummond 2009), and since the mid-2000s, A-level Computing syllabi have placed more emphasis on programming than A-level ICT (AQA 2009; OCR 2009). Recently, more students are applying and entering university with a BTEC qualification, but CSEd literature has yet to establish the usefulness of BTEC IT and Games qualifications for preparing students for undergraduate study. This finding informed a direction of the primary research, resulting in a comparison of L4 programming performance of BTEC and A-level entrants, as reported in Chapter 4.

More generally, the unprecedented increase in HE participation - from approximately 5% of all school-leavers in the 1950s to some 50% of school-leavers in the 2010s (Figure 1.1) – has led to the diversification of student attitudes towards HE. More students now concentrate on their potential career prospects (High Fliers Research 2015) and more undertake part-time work alongside studying to fund their living expenses (Watson et al. 2014). The review of literature also identified that those applying the surface learning approach tended to have ‘performance goals’, whereas those applying the deep learning approach tended to have ‘mastery goals’. Furthermore, studies generally reported that the deep learning approach had a stronger correlation with programming performance than the surface learning approach.

Investigation of learner profiles and personality traits revealed that CS students tend to lean toward abstract and thinking preferences. Success in programming courses also links with correct understanding and mental models of abstract programming concepts. However, studies indicate that poor cognitive control (excessive procrastination and a tendency to be easily distracted by social media) and the inability to concentrate on learning materials for sustained periods of time, all contribute to poor performance. Although one study indicates that occasional and infrequent interference of non-academic tasks are not as detrimental to performance as frequent interruption (Rosen et al. 2011).

Learning and teaching techniques which encourage the development of social bonds and collaboration were found to improve retention, engagement and programming grades. Students reported that they found it easier to concentrate when working in groups as they encouraged each other to stay ‘on task’ if they noticed signs of distraction. This aligns with other research which found that collaborative modes enabled quicker completion of tasks. Teague (2011) recommended the utilisation of video and audio recording to investigate the development of novice programmers over a typical introductory programming semester. This informed the research methods applied in Chapter 5 in an attempt to understand typical classroom behaviours and how they influence programming performance.

The identification of effective pedagogical interventions from literature (Objective 2)

The review of typical programming course elements discussed the contrasting views of those who lean toward novice friendly methods (involving high level programming languages, simple development environments, and small-scale exercises) and those who wish to integrate the languages and complex environments used in industry to address large-scale ‘real-life’ problems.

Conventional materials (such as slides, textbooks and lecture materials) do not tend to explicitly teach the problem-solving processes (Bennedsen & Caspersen 2005) which are required in the workplace. This finding informed the inclusion of ‘demonstration’ sessions in the ‘incremental’ teaching model (Chapter 6) which aimed to articulate the processes and approaches typically applied to problems.

Bennedsen and Caspersen (2007) discovered that small class sizes (less than 30) outperformed larger classes (typically 100 or more). The review of four active learning techniques (Peer Instruction, Pair Programming, use of Worked Examples, and Games-Based Learning) further illustrated the importance of peer learning and collaborative contexts. The implementation of these techniques that encourage active learning retained far more students compared with conventional lecture orientated delivery modes (Freeman et al. 2014). However, uptake of such models outside of institutions involved in studies is often restricted by lack of time (Barker et al. 2015), or concerns regarding a lack of supporting evidence (Perrotta et al. 2013).

The review also discussed the three main assessment formats (small-scale exercises, projects/assignments and written examinations) and established a need for more formative assessment. This finding, along with earlier exploratory work, informed the introduction and evaluation of unassessed quizzes, as discussed in Chapter 6.

Comparison of learning performance of A-level and BTEC entrants (Objective 3)

The comparison between the L4 programming performance of BTEC and A-level entrants extends previous work that was concerned solely with the performance of A-level ICT and A-Level Computing entrants (Boyle et al. 2002; Drummond 2009). This research contributes evidence that A-level students significantly outperformed BTEC students (by an entire grade) in an L4 introductory programming course (see Figure 4.2). This difference is interesting for two reasons: (1) the UCAS points of A-level entrants were weakly correlated with L4 programming grade in contrast to the moderately strong correlation between L4 grade and the UCAS points of BTEC entrants; and (2) most A-level students in the sample (75%) did not take A-level ICT or A-level Computing.

The lack of correlation between L4 programming grade and code knowledge (Section 5.1) is further evidence that prior programming knowledge at L3, or extent of knowledge at L4, does not appear to be a crucial requirement for successfully passing L4 assignments involving small-scale exercises.

Distinction between programmer dialogue that informs understanding of programming concepts and dialogue concerned with completing tasks (Objective 4)

Analysis of video recordings demonstrated that learners can develop a correct and working solution despite considerable variation in their understanding of the programming concepts and the problem-solving approaches employed. Participants mostly adopted 'trial and error' approaches toward repairing programs, and few pairs attempted to fully understand task requirements before coding their solutions. These findings suggest that coursework was not a sensitive indicator of true programming ability, hence the poor correlation between programming grade and ability.

In addition, recordings reveal that undergraduate novice programmers demonstrate behaviours consistent with the 'Piagetian' stages of development (Lister 2016; Teague 2015). Whilst the original theory concerns the cognitive development of children (Piaget 1952), Teague (2015) and Lister (2016) suggest that the adaptation of the theory (Neo-Piagetian theory) could be used to inform knowledge creation of novice programmers (epistemology). However, findings are also consistent with a 'social' constructivist theory of learning. A case study of Games Development groups revealed the impact that a sense of belonging and peer teaching can have on engagement, retention, learning progress and learning autonomy. Relatively experienced programmers would act as tutors and mentors asking specific questions with the aim of helping their more inexperienced peers to think differently. This social context was found to encourage dialogue that concentrated on understanding concepts to a greater extent than Pair Programming sessions.

Effective pedagogical interventions and the 'Incremental' teaching approach (Objective 5)

The implementation of periodic summative assessment revealed that non-participation with, or failure to pass the first submission point is a reliable indicator of vulnerability to pass an introductory L4 programming course. Furthermore, this research discovered a moderately strong correlation between the performance in unassessed quizzes (given in Week 4) and final L4 programming grade. Such findings suggest that engagement and performance in these early assessed or unassessed activities should be closely monitored for identifying potentially vulnerable individuals.

Chapter 6 also recommends an 'incremental' approach to teaching programming concepts. Figure 6.4 demonstrates how conventional one-hour lecture and two-hour practical sessions can be divided into shorter cycles of presenting new information, followed by a relevant exercise, and then a full demonstration, with discussion, of how working solutions may be developed.

This work also responds to concerns regarding a deficit of CSEd research underpinned by educational theory (Fincher & Petre 2004; Gordon 2016; Ihantola et al. 2015: 44), by relating observations and findings to constructivism (Vygotsky 1976), cognitive load theory (Sweller 2011) and observed attention spans (Davidson 2011; Rosen & Gazzaley 2016). The implementation of the 'incremental teaching model' led to increased attentiveness and engagement with class materials. L4 students subjected to this teaching model also retained more programming knowledge at the end of the course than previous academic years.

This teaching model can be applied in both traditional and contemporary contexts, thereby responding to the needs of academics with limited time, and or constrained by QA processes that often restrict the type and extent of changes that may be made to courses (Barker et al. 2015; Ni 2009). The model is equally adaptable to contemporary and non-conventional formats of HE delivery (such as two-year degrees, degree-apprenticeships and distance learning). Although components of the model apply specifically to the programming process (providing a programming exercise followed by demonstrating the solution), this model could be adapted to deliver related subjects including STEM (science, technology, engineering and mathematics) topics. Thus, it would be relatively straightforward to replace programming exercises with mathematical topics, and for instructors to deliver tasks and demonstrate corresponding processes for solving problems.

7.3 Limitations and future recommendations

A-levels and BTECs – the need for improved sample representation of UK and overseas institutions, evaluate new syllabi, and address disproportionate numbers of men to women

Attempts to compare the undergraduate performance of entrants with A-level Computing and BTEC Games qualifications were limited by small sample sizes. Although the research intention was to be representative of undergraduate programming students as a whole, resources were only sufficient to conduct a study involving multiple cohorts from one UK institution and one other HE organisation in Sri Lanka. Nevertheless, some confidence may be attached to the consistency of findings across the three years of study. Although results from data analyses for a college in Sri Lanka were consistent with UK findings, further investigations would be strengthened by stratifying samples to be representative of undergraduates from other UK and overseas institutions.

Following the introduction of a new BTEC IT syllabus (Pearson 2016c), there is a need to establish whether this is more or less effective in preparing students for undergraduate programming courses than the previous BTEC IT curriculum. It would also be useful to investigate the degree of programming coverage in the new BTEC Computing (Pearson 2016b) and A-level Computer Science qualifications (AQA 2019; OCR 2018), and whether the shortages of capable teaching staff with programming expertise (Johnson 2013; Savage 2018) have been addressed.

Furthermore, there is a need to investigate whether the introduction of examinations to BTEC National awards and stricter rules regarding resubmission of work counting toward L2 and L3 (Ofqual 2018: 57) have addressed the concerns regarding BTEC grade inflation (HEFCE 2015). A recent report suggests that further work is also required to attract and engage more young women in A-level Computing (Cellen-Jones 2017).

Video analysis, collaboration and group culture – towards the development of more representative assessment and measurement of learning (useful for informing research methods)

Regrettably, due to low uptake by students, this research was unable to fully adopt Teague's (2011) suggested research design for recording student development over an entire semester. Whilst the analysis of standalone Pair Programming episodes (Section 5.3) did reveal important behaviours for explaining patterns of high attainment amongst collaborative students, implementing Teague's (2011) original research design could reveal further important insights regarding how learners develop over the duration of an introductory course. Even though some platforms are now able to record data relating to the development of code and interaction with the IDE, and noting that some will even make precise logs of keystrokes (Ihantola et al. 2015), these do not capture data pertaining to thinking processes and dialogue between learners. Interpretation of these latter aspects is essential for more completely understanding the role of social and collaborative contexts in learning development.

Formal adoption of the Pair Programming method by L4 programming courses would be more representative of the workplace and would then better prepare students for teamwork in industry. However, a productive line of inquiry for future research would be to determine how the Pair Programming format can be adapted to ensure that programmers develop their understanding as they complete tasks. Further to the findings reported in Section 5.3, future research might also consider how tutors can facilitate and cultivate a dialogue between peers that best promotes deeper learning of concepts and more structured problem-solving processes.

Since summative assessment does not adequately measure the variety of problem-solving approaches employed and the cognitive level of programmers, analysis of programmer dialogue (Teague 2011; Teague 2015) may provide a more realistic means for evaluating the development of skills and understanding.

The design of assessment strategies and of teaching materials

Research findings were consistent with those of earlier studies that concluded that teaching content and assessments may be written and communicated in a language style that may be too advanced for the cognitive level of its intended audience (Luxton-Reily et al. 2017; Teague 2015). In addition, assessment (from exam questions, to projects, to small-scale lab exercises) commonly requires learners to apply several programming concepts simultaneously (Luxton-Reily et al. 2017). Following the specific difficulties that learners experience with iterative and array statements (Teague 2015), future research may wish to evaluate strategies to more effectively teach these concepts.

The provision of full algorithms (with the intention that these are to be subsequently translated to code) may be more appropriate for novice programmers before they progress to developing solutions from high-level descriptions of task requirements. Future research should investigate whether the use of algorithms can mitigate the frustration and confusion of not knowing how to approach a problem. This could help demonstrate that, contrary to popular belief, programming is not ‘hard’ but achievable if the correct steps are followed (Biggs & Tang 2011). Appropriate ‘fading’ of full algorithms throughout the course may also encourage transition from lower-order to higher-order thinking.

Collaboration between CS educators and researchers from different institutions to develop, review and disseminate learning materials suitable for the cognitive abilities of novice programmers has the potential to improve CS education. However, there are a number of barriers toward centralising research and learning materials (Lister 2005b); mainly time (Barker et al. 2015), but also the accessibility of research and learning materials – as CSEd research is published across a wide range of journals (Fincher & Petre 2004).

Flexible learning modes, online learning and data analytics toward creating dynamic predictors

Some proprietary online platforms and modern IDEs are currently able to record data relating to student access and engagement with teaching material, exercises and assessments (Brown & Altadmri 2015; Croft & England 2019). Collection and analysis of large datasets, also referred to as 'data mining' or 'data analytics', can produce 'dynamic predictors' (Ahadi et al. 2015; Ihantola et al. 2015; Watson et al. 2013; Watson et al. 2014) that forecast and model the complexities of grade performance with significantly greater accuracy than 'static' predictors ('static' referring to measurements taken at one point in time). Coupled with recent progress toward automated marking and adaptive assessment (Croft & England 2019; Vaughan 2017), the development of a 'dynamic' system could enable students to learn at their own pace, thereby removing the constraints of requiring classes to progress in time with the instructor.

Although time and resource limitations are well documented (Barker et al. 2015), HE institutions may be required to adopt flexible delivery modes to attract applicants in a rapidly changing and increasingly competitive market (House of Commons Education Select Committee 2018: 13; Woodgates 2018).

References

Abdul-Rahman, S. S., and du Boulay, B. (2013) ‘Learning programming via worked-examples: Relation of learning styles to cognitive load’. *Computers in Human Behavior* 30, 286-298

Adams, R. (2015) *First crop of £9,000 tuition fee-paying UK graduates 'more focused on pay'* [online] available from <http://www.theguardian.com/education/2015/jun/22/9000-tuition-fee-graduates-focused-pay-high-fliers> [December 2015]

Adey, P., and Shayer, M. (1994) *Really Raising Standards: Cognitive Intervention and Academic Achievement*. London: Routledge

Ahadi, A., and Lister, R. (2013) ‘Geek Genes, Prior Knowledge, Stumbling Points and Learning Edge Momentum: Parts of the One Elephant?’. In *Proceedings of the ninth annual international ACM conference on International computing education research* 124-128

Ahadi, A., and Mathieson, L. (2019a) ‘A Comparison of Three Popular Source code Similarity Tools for Detecting Student Plagiarism’. *In Proceedings of the Twenty-First Australasian Computing Education Conference (ACE '19)* 112-117

Ahadi, A., Lister, R., Haapala, H., and Vihavainen, A. (2015) ‘Exploring Machine Learning Methods to Automatically Identify Students in Need of Assistance’. *In Proceedings of the eleventh annual International Conference on International Computing Education Research (ICER '15)* 121-130

Allert, J. (2004) ‘Learning style and factors contributing to success in an introductory computer science course’. *Fourth IEEE International Conference on Advanced Learning Technologies (ICALT’04)* 385-389

Alley, S., and Smith, M. (2004) *Timeline: tuition fees* [online] available from <http://www.theguardian.com/education/2004/jan/27/tuitionfees.students> [November 2015]

Almstrum, V. L., Hazzan, O., Guzdial, M., and Petre, M. (2005) ‘Challenges in computer science education research’. *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education (SIGCSE ’05)* 191-192

Alvarado, C., and Dodds, Z. (2010) ‘Women in CS: an evaluation of three promising practices’. In *Proceedings of the 41st ACM technical symposium on Computer Science education, SIGSCE’10.* [online] 57-61. available at: https://www.cs.hmc.edu/~alvarado/papers/fp068-alvarado.pdf> [September 2017]

Anderson, J. A. (2005) *Cognitive Psychology and its implications* (6th ed.) NY: Worth Publishers

AQA (2009) *GCE Computing Specification for AS exams 2009 onwards and A2 exams 2010 onwards (version 1.5)* [online] available from <https://www.thestudentroom.co.uk/attachment.php?attachmentid=98784&amp;d=1301852145> [February 2017]

AQA (2014a) *GCE AS and A Level Specification Computing* [online] available from <http://filestore.aqa.org.uk/subjects/specifications/alevel/AQA-2510-W-SP-14.PDF> [February 2017]

AQA (2014b) *GCE AS and A Level Specification: Information and Communication Technology* [online] available from <http://filestore.aqa.org.uk/subjects/specifications/alevel/AQA-2520-W-SP-14.PDF> [February 2017]

AQA (2019) *AS and A Level Computer Science* [online] available from <https://filestore.aqa.org.uk/resources/computing/specifications/AQA-7516-7517-SP-2015.PDF> [June 2019]

Ariga, A., and Lleras, A. (2011) 'Brief and Rare Mental 'Breaks' Keep You Focused. Deactivation and Reactivation of Task Goals Pre-empt Vigilance Decrements'. *Cognition* 118 (3), 439-443

Association for Computing Machinery (2013) *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science* [online] available from <https://www.acm.org/binaries/content/assets/education/cs2013_web_final.pdf> [December 2017]

Ateeq, M., Habib, H., Umer, A., and Rehman, M. U. (2014) 'C++ or Python? Which One to Begin with: A Leaner's Perspective'. In *Proceedings of International Conference on Teaching and Learning in Computing and Engineering (LaTiCE '14)* 64-69

Atkinson, R. K., Derry, S. J., Renkl, A. and Wortham, D. (2000) 'Learning from Examples: Instructional Principles from the Worked Examples Research'. *Review of Educational Research* 70 (2), 181-214

Atkinson, R. K., Renkl, A., and Merrill, M. M. (2003) 'Transitioning from studying examples to solving problems: effects of self-explanation prompts and fading worked-out steps'. *Journal of Educational Psychology* 95 (4), 774-783

Avila, P. (2016) *My Experience with Pair Programming* [online] available from <https://dev.to/raulavila/my-experience-with-pair-programming> [June 2017]

Azalov, P., Azaloff, S., and Zlatarova, F. (2004) 'Work in Progress - Comparing Assessment Tools in Computer Science Education: Empirical Analysis'. *Proceedings of 34th ASEE/IEEE Frontiers in Education Conference* 2, 18-19

Bagley, C. A. and Chou, C. C. (2007) 'Collaboration and the Importance for Novices in Learning java Computer Programming'. *ITiCSE Conference '07* 171-175

Barker, L., Hovey, C. L., and Gruning, J. (2015) 'What influences CS faculty to adopt teaching practices?' In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education SIGCSE '15* 604-609

Barker, L. J., Garvin-Doxas, K., and Jackson, M. (2002) 'Defensive climate in the computer science education'. In D. Knox, editor, *The Proceedings of the Thirty-third SIGCSE Technical Symposium on Computer Science Education* [online] 43-47. available from <http://inst.eecs.berkeley.edu/~cs301/fa13/cs375/p43-barker.pdf> [September 2017]

Barker, R. J. and Unger, E. A. (1983) 'A predictor for success in an introductory programming class based upon abstract reasoning development'. *ACM SIGCSE Bulletin* 15 (1), 154-158

Barks, H., Searight, R., and Ratwik, S. (2011) 'Effects of text messaging on academic performance'. *Journal of Pedagogy and Psychology "Signum Tempori*s" 4 (1), 4–9

Barnes, T., Richter, H., Chaffin, A., Godwin, A., Powell, E., Ralph, T., Matthews, P., and Jordan, H. (2007) 'Game2Learn: A study of games as tools for learning introductory programming concepts'. *(SIGCSE'07)* [online] 121-125. available from <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.470.3491&rep=rep1&type=pdf> [January 2019]

Bathmaker, A. M. (2003) 'The Expansion of Higher Education: A Consideration of Control, Funding and Quality'. In Bartlett, S. and Burton, D. (eds) *Education Studies. Essential Issues*, London: Sage. [online] 169-189. available from <http://eprints.uwe.ac.uk/131/1/Bathmaker_-_the_expansion_of_higher_education_chapter_prepub.doc> [January 2019]

Beck, K. (2000) *Extreme Programming explained: embrace change*. Addison Wesley: England. 30-40

Beck, L. L., Chizhik, A. W., and McElroy, A. C. (2005) 'Cooperative Learning Techniques in CS1: design and experimental evaluations'. *SIGCSE '05: Proceedings of the 36th SIGCSE technical symposium on Computer science education* [online] 470-474. available from <http://delivery.acm.org/10.1145/1050000/1047495/p470-beck.pdf?ip=79.69.76.228&id=1047495&acc=OPEN&key=4D4702B0C3E38B35%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35%2E6D218144511F3437&__acm__=1551902759_c721accd46d05f5492196b3cfe3a91d7> [September 2017]

Bednarik, R., and Franti, P. (2004) 'Survival of students with different learning preferences'. *Proceedings of the Seventh Baltic Sea Conference on Computing Education Research (Koli Calling 2004)* 121-125

Begel, A., and Nagaan, N. (2008) 'Pair Programming: what's in it for me?'. *ESEM '08: Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement* 120-128

Ben-Ari, M. (1998) 'Constructivism in computer science education'. *Journal of Computers in Mathematics and Science Teaching* 257-261

Ben-Bassat Levy, R., Ben-Ari, M., and Uronen, P. A. (2003) 'The Jeliot 2000 program animation system'. *Computers & Education* [online] 40 (1), 15-21. available from < https://s3.amazonaws.com/academia.edu.documents/3441717/ben-bassat-levy2003.pdf?response-content-disposition=inline%3B%20filename%3DThe_Jeliot_2000_Program_Animation_System.pdf&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWOWYYGZ2Y53UL3A%2F20190706%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20190706T162749Z&X-Amz-Expires=3600&X-Amz-SignedHeaders=host&X-Amz-Signature=06b8c0846054dfb18557a9c889c868f0233a816f42c0f99d76e287c9a68d7f7f> [October 2015]

Bennedsen, J., and Caspersen, M. E. (2007) 'Failure rates in introductory programming'. *SIGCSE Bulletin* [online] 39 (2), 32–36. available from <http://users-cs.au.dk/mic/publications/journal/25--bulletin2007.pdf> [May 2017]

Bennedsen, J., and Caspersen, M. E. (2005) Revealing the programming process. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education* 186-190

Bennedsen, J., and Caspersen, M. E. (2008) 'Optimists have more fun, but do they learn better? On the influence of emotional and social factors on learning introductory computer science'. *Computer Science Education* 18 (1), 1-16

Bennedsen, J., and Caspersen, M. E. (2005) ‘An Investigation of Potential Success Factors for an Introductory Model-Driven Programming Course’. *Proceedings of ICER’05* 155-163

Bergin, S., and Reilly, R. (2005a) ‘Programming: factors that influence success’. *SIGCSE Bulletin* 37 (1), 411-415

Bergin, S., and Reilly, R. (2005b) ‘The influence of motivation and comfort level on learning to program’. *Proceedings of the 17th Workshop of the Psychology of Programming Interest Group (IG 05)* [online] 293-304. available from <http://eprints.teachingandlearning.ie/1770/2/Bergin%20and%20Reilly%202005.pdf> [December 2017]

Berglund, A., Daniels, M., and Pears, A. (2006) ‘Qualitative Research Projects in Computing Education Research: An Overview’. *In Proceedings of Eighth Australasian Computing Education Conference (ACE2006)* [online] 52, 25-33. available from <https://pdfs.semanticscholar.org/b2a8/7855c94cbfaa12c2ca2e2d1820a15f3b87e0.pdf> [April 2019]

Berglund, A., Kinnunen, P., and Malmi, L. (2007) ‘A doctoral course in research methods in computing education research: how should we teach it?’ In *Proceedings of the Seventh Baltic Sea Conference on Computing Education Research (Koli Calling 2007)* 88, 175-178

Bevan, J., Werner, L., and McDowell, C. (2002) ‘Guidelines for the use of Pair Programming in a freshman programming class’. In *Proceedings from 15th Conference on Software Engineering Education and Training* 100

Biggs, J., and Tang, C. (2011) *Teaching for Quality Learning at University*. OUP: McGraw Hill

Biggs, J., Kember, D., and Leung, D. (2001) ‘The Revised Two Factor Study Process Questionnaire: R-SPQ-2F’. *British Journal of Educational Psychology* 71, 133-149

Biggs, J. (1987) *Student approaches to learning and studying*. Victoria: Australian Council for Educational Research

Bonar, J. and Soloway, E. (1985) ‘Preprogramming knowledge: a major source of misconceptions in novice programmers’. *Human-Computer Interaction* 1, 133-161

Booth, J., Lange, K. E, Koedinger, K. R., and Newton, K. J. (2013) ‘Example Problems That Improve Student Learning in Algebra: Differentiating between Correct and Incorrect Examples’. *Learning and Instruction* 25, 24–34

Booth, J., Cooper, L. A., Donovan, M. S., Huyghe, A., Koedinger, K. R., and Paré-Blagoev, E. J. (2015) ‘Design-Based Research within the Constraints of Practice: Algebra By Example’. *Journal of Education for Students Placed at Risk* 20, 79–100

Bornat, R. (2011) *Some problems of teaching (learning) first-year programming (plus a glimmer of hope)* [online] available from <http://www.researchgate.net/profile/Richard_Bornat/publication/266342475_Some_problems_of_teaching_%28learning%29_first-year_programming_%28plus_a_glimmer_of_hope%29/links/54b79bd20cf24eb34f6ebfbf.pdf> [July 2015]

Bornat, R. (2014) *Camels and Humps: a retraction* [online] available from <http://www.eis.mdx.ac.uk/staffpages/r_bornat/papers/camel_hump_retraction.pdf> [July 2015]

Bornat, R., Dehnadi, S., and Simon (2008) ‘Mental models, consistency and programming aptitude’. *Proceedings of the Tenth Australasian Computing Education Conference* (*ACE 2008*) 53–62

Borstler, J., Nordstrom, M., Westin, L. K., Mostrom, J. E., and Eliasson, J. (2008) ‘Transitioning to OOP/Java – A Never Ending Story’. In J. Bennedsen et al. (2008) *Teaching of Programming*, 2008. Springer: Berlin

Bounds, A., and O’Connor, S. (2015) *Digital economy transforms UK workforce* [online] available from <https://www.ft.com/content/5ac2e590-c741-11e4-9e34-00144feab7de> [October 2016]

Boustedt, J., Eckerdal, A., McCartney, R., Mostrom, J. E., Ratcliffe, M., Sanders, K., and Zander, C. (2007) ‘Threshold Concepts in Computer Science: Do they exist and are they useful?’ *SIGCSE Bulletin inroads* [online] 39 (1), 504-508. available from <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.630.1504&rep=rep1&type=pdf> [July 2016]

Bouwkamp, K. (2016) *The 9 Most In-Demand Programming Languages of 2016* [online] available from <http://www.codingdojo.com/blog/9-most-in-demand-programming-languages-of-2016/> [September 2017]

Boyle, R. D., Carter, J. E., and Clark, M.A.C. (2002) ‘What Makes Them Succeed? Entry, progression and graduation in Computer Science’. *Journal of Further and Higher Education* 26 (1), 3-18

Braught, G., MacCormick, J., et al. (2010) ‘The Benefits of Pairing by Ability’. *41st ACM technical symposium on Computer Science Education* 249-253

Braught, G., Wahls, T., and Eby, M., et al. (2008) ‘The Effects of Pair-Programming on Individual Programming Skill’. *39th SIGCSE technical symposium on Computer Science education (SIGCSE ’08)* 200-204

British Chambers of Commerce (2017) *BCC: Shortage of digital skills hampering business productivity and growth* [online] available from <http://www.britishchambers.org.uk/press-office/press-releases/bcc-shortage-of-digital-skills-hampering-business-productivity-and-growth.html> [June 2017]

Brooks, J. H., and DuBois, D. L. (1995) ‘Individual and Environmental Predictors of Adjustment During the First Year of College’. *Journal of College Student Development* 36 (4), 347-360

Broughton, N. (2013) *In the Balance: The STEM Human Capital Crunch* [online] available from <http://www.smf.co.uk/wp-content/uploads/2013/03/Publication-In-The-Balance-The-STEM-human-capital-crunch.pdf> [May 2017]

Brown, A. L. (1992) ‘Design experiments: theoretical and methodological challenges in creating complex interventions in classroom settings’. *The Journal of the Learning Sciences* 2 (2), 141-178

Brown, N., and Altadmri, A. (2014) ‘Investigating novice programming mistakes: educator beliefs vs student data’. In *Proceedings of the tenth annual conference on International computing education research* 43-50

Brown, N., and Altadmri, A. (2015) ‘37 Million Compilations: Investigating Novice Programming mistakes in large-scale student data’. *SIGCSE ’15* 522-527

Browne, J. (2010) *Securing a sustainable future for higher education: an independent review of higher education funding & student finance* [online] available from <https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/422565/bis-10-1208-securing-sustainable-higher-education-browne-report.pdf> [April 2017]

Bryman, A. (2007) 'Barriers to integrating quantitative and qualitative research'. *Journal of Mixed Methods Research* 1 (1), 8-22

Brynjolfsson, E., and McAfee, A. (2014) *The Second Machine Age: Work, Progress, and Prosperity in a Time of Brilliant Technologies*. W. W. Norton & Company: New York

Bucks (2015a) *Module Board Report [CO452 2014-2015].* [unpublished]

Bucks (2015b) *Buckinghamshire New University Access Agreement 2015-16* [online] available from <https://www.offa.org.uk/agreements/Buckinghamshire%20New%20University.pdf> [September 2016]

Bucks (2015b) *Module Board Report [CO453 2014-2015]*. [unpublished]

Bucks (2016a) *Module Board Report [CO452 2015-2016]*. [unpublished]

Bucks (2016b) *Module Board Report [CO453 2015-2016]*. [unpublished]

Bureau of Labor Statistics (2014) *STEM 101: Intro to tomorrow's jobs. Occupational Outlook Quarterly, Spring 2014* [online] available from <https://www.bls.gov/careeroutlook/2014/spring/art01.pdf> [May 2017]

Byrne, P., and Lyons, G. (2001) 'The effect of student attributes on success in programming.' In *Proceedings of the ACM Annual Conference on Innovation and Technology in Computer Science Education* [online] 49-52 available from <https://www.researchgate.net/publication/220807520_The_Effect_of_Student_Attributes_on_Success_in_Programming> [September 2017]

Cantero, J. V. (2018) *Toward better error messages in Scalac* [online] available from <https://contributors.scala-lang.org/t/towards-better-error-messages-in-scalac/1470> [July 2019]

Carroll, W. M. (1994) 'Using Worked Examples as an Instructional Support in the Algebra Classroom'. *Journal of Educational Psychology* 86 (3), 360–67

Carter, A., and Bathmaker, A. M. (2017) 'Prioritising progression over proficiency: limitations of teacher-based assessment within technician-level vocational education'. *Journal of Further and Higher Education* 41 (4), 460-474

Carter, J., and Jenkins, T. (2001) 'Where have all the girls gone? What entices female students to apply for a computer science degree'. In *Proceedings of 2nd Annual LTSN for Information and Computer Science Conference,* London, UK, Learning and Teaching Support Network for Information and Computer Science 72-77

Carver, S. M. (1986) *Transfer of LOGO Debugging Skill: Analysis, Instruction, and Assessment*. PhD dissertation. Carnegie Mellon University: Pittsburgh, PA

Caspersen, M. E., and Bennedsen, J. (2007) 'Instructional design of a programming course: a learning theoretic approach'. In *Proceedings of the third international workshop on Computing education research (ICER '07)* 111-122

Caspersen, M. E., and Kölling, M. (2009) ‘STREAM: A First Programming Process’. *ACM Transactions on Computing Education* 9 (1), 4:1-29

Caspersen, M. E., Larsen, K. D., and Bennedsen, J. (2007) ‘Mental Models and programming aptitude’. In *ITiCSE ’07: Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education* 206-210

Caspersen, M. E., Borstler, J., Decker, A. and Alphonce, C. (2008) ‘Worked Examples for Sound Object-Oriented Pedagogy: A “killer” workshop’. *OOPSLA ’08* 777-778

Cellan-Jones, R. (2011) *Coding: the new Latin* [online] available from <http://www.bbc.co.uk/news/technology-15916677> [September 2014]

Cellan-Jones, R. (2014) *A Computing revolution in schools* [online] available from <http://www.bbc.co.uk/news/technology-29010511> [September 2014]

Cellan-Jones, R. (2017) *Computing in schools - alarm bells over England's classes* [online] available from <http://www.bbc.co.uk/news/technology-40322796> [October 2017]

Chambers, J. M., Cleveland, W. S., Tukey, P. A., and Kleiner, B. (1983) ‘Comparing Data Distributions’. In *Graphical Methods for Data Analysis* 62

Clancy, M. J., and Linn, M. C. (1990) ‘Functional Fun’. In *Proceedings of the 21st SIGCSE Technical Symposium on Computer Science Education*, published as *SIGCSE Bulletin* 22 (1), 63-67

Clark, M., and Boyle, R. D. (2005) ‘The Transition from School to University: Would Prior Study of Computing Help? Computer Literacy to Informatics Fundamentals’. *International Conference on Informatics in Secondary Schools -- Evolution and Perspectives (ISSEP 2005)* 37-45

Clark, R. C., Nguyen, F., and Sweller, J. (2011) *Efficiency in Learning: Evidence-based Guidelines to Manage Cognitive Load*. New York: John Wiley & Sons

Clark, R., Nguyen, F., and Sweller, J. (2006) *Efficiency in learning: evidence-based guidelines to manage cognitive load*. San Francisco, CA: John Wiley & Sons, Inc

Code.org (2016) *The largest learning event in history: hour of code* [online] available from <https://hourofcode.com/gb/files/hoc-one-pager.pdf> [Jan 2019]

Coffield, F., Moseley, D., Hall, E., and Ecclestone, K. (2004a) *Learning Styles and Pedagogy in Post-16 Learning: A Systematic And Critical Review* [online] available from <http://sxills.nl/lerenlerennu/bronnen/Learning%20styles%20by%20Coffield%20e.a..pdf> [July 2015]

Coffield, F., Moseley, D., Hall, E., and Ecclestone, K. (2004b) *Should we be using learning styles? What research has to say to practice. Learning&Skills research centre* [online] available from <http://www.itslifejimbutnotasweknowit.org.uk/files/LSRC_LearningStyles.pdf> [July 2015]

Cohen, L., Manion, L., and Morrison, K. (2011) (7th ed) *Research Methods in Education*. Routledge: Taylor Francis

Cohoon, J., and Aspray, W. (2006) ‘A critical review of the research on women’s participation in postsecondary computing education’. In *Women and information technology: Research on underrepresentation*. MIT Press: Cambridge, 137-180

Colby, J. (2004) ‘Attendance and attainment’. *Fifth Annual Conference of the Information and Computer Sciences—Learning and Teaching Support Network (ICS-LTSN).* [online] available from <http://www.ics.heacademy.ac.uk/italics/Vol4-2/ITALIX.pdf> [May 2017]

CompTIA (2017a) *CompTIA UK IT Employment Snapshot 2016 Q4* [online] available from <https://www.comptia.org/resources/comptia-uk-it-employment-snapshot-2016-q4> [May 2017]

CompTIA (2018a) *CompTIA UK IT Employment Snapshot 2017 Q4* [online] available from <https://www.comptia.org/resources/comptia-uk-it-employment-snapshot-2017-q4> [Jan 2019]

CompTIA (2018b) *CompTIA UK IT Employment Snapshot 2018 Q3* [online] available from <https://www.comptia.org/resources/comptia-uk-it-employment-snapshot-2018-q3> [Jan 2019]

Computing at School (2012) *Computer Science as a school subject: Seizing the opportunity* [online] available from <http://www.computingatschool.org.uk/data/uploads/Case%20for%20Computing.pdf> [November 2015]

Computing at School and NAACE (2014) *Computing in the national curriculum: A guide for secondary teachers* [online] available from <http://www.computingatschool.org.uk/data/uploads/cas_secondary.pdf> [December 2015]

Constantine, L. L. (1995) *Constantine on Peopleware*. Yourdon Press: Englewood Cliffs, NJ

Cooper, S., Dann, W., and Pausch, R. (2000) 'Alice: a 3-D tool for introductory programming concepts'. In: *Proceedings of the 5th Annual CCSC Northeastern Conference*, 107-116

Cooper S., Forbes J., Fox A., Hambrusch S., Ko A., and Simon B. (2016) *The Importance of Computing Education Research: A white paper prepared for the Computing Community Consortium committee of the Computing Research Association* [online] available from <https://cra.org/ccc/wp-content/uploads/sites/2/2015/01/CSEdResearchWhitePaper2016.pdf> [April 2019]

Corbett, A. T., and Anderson, J. R. (2001) 'Locus of feedback control in computer-based tutoring: Impact on learning rate, achievement and attitudes'. In *Proceedings of the SIGCHI conference on Human factors in computing systems* 245-252

Corney, M., Lister, R., and Teague, D. (2011) 'Early Relational Reasoning and the Novice Programmer: Sweaing as the Hello World of Relational Reasoning'. In *Proceedings of Thirteenth Australasian Computing Education Conference (ACE 2011)* [online] 95-104. available from: <http://crpit.com/confpapers/CRPITV114Corney.pdf> [June 2015]

Corsten, L. C. A., and Gabriel, K. R. (1976) 'Graphical exploration in comparing variance matrices'. *Biometrics* 39, 159-168

Coughlan, S. (2010) *Students face tuition fees rising to £9,000* [online] available from <http://www.bbc.co.uk/news/education-11677862> [December 2015]

Council of Professors and Heads of Computing (2012) *CS Graduate Unemployment Report 2012* [online] available from <http://cphcuk.files.wordpress.com/2013/12/cs_graduate_unemployment_report.pdf> [April 2017]

Council of Professors and Heads of Computing (2016) *Computing Graduate Employability: Sharing Practice* [online] available from <https://cphcuk.files.wordpress.com/2016/01/computinggraduateemployabilitysharingpractice.pdf> [May 2017]

Creswell, J. W., and Plano Clark, V. L. (2011) (2nd ed) *Designing and Conducting Mixed Methods Research.* Sage: London

Croft, D., and England, M. (2019) 'Computing with Codio at Coventry University: Online virtual Linux boxes and automated formative feedback'. *Proceedings of the 3rd Conference on Computing Education Practice (CEP '19)* 16: 1-4

Crouch, C. H. Watkins, J., Fagan, A. P., and Mazur, E. (2007) 'Peer Instruction: Engaging students one-on-one, all at once'. In Redish, E. F., and Cooney, P. J. (Eds.), *Research-based reform of university physics.* College Park, MD: American Association of Physics Teachers

Crouch, C. H., and Mazur, E. (2001) 'Peer Instruction: Ten years of experience and results'. *American Association of Physics Teachers* [online] 69 (9), 970-977. available from <http://web.mit.edu/jbelcher/www/TEALref/Crouch_Mazur.pdf> [October 2017]

Curtis, P. (2009) *Universities warn of stiff competition as recession prompts big rise in applications* [online] available from <https://www.theguardian.com/education/2009/feb/16/university-alications-recession> [January 2019]

Cuthbert, P. F. (2005) 'The student learning process: Learning styles or learning approaches?'. *Teaching in Higher Education* 10: 235-249

Dalton, H., and MacKay, D. (2019) 'BTEC Students into Higher Education: Listening to Learner's Voices'. [presentation slides] NEON Summer Symposium 2019, June 2019 [online] available from <https://www.educationopportunities.co.uk/wp-content/uploads/Session-2-David-MacKay-Head-of-Stakeholder-Relationships-Hayley-Dalton-Head-of-Vocational-Design-Pearson.pdf> [January 2020]

Davidson, C. N. (2011) *Now You See It: How the brain science of attention will transform the way we live, work, and learn.* Viking: New York

De Kock, E., and Gardner, T. (2015) *Code Clubs for Kids – UK Edition* [online] available from <https://www.techagekids.com/2015/08/code-clubs-for-kids-uk-edition.html> [January 2019]

de Raadt, M., and Simon (2011) 'My Students Don't Learn the Way I Do'. *Proceedings of the Thirteenth Australasian Computing Education Conference (ACE 2011)* 114, 105-112

de Raadt, M. (2008) *Teaching Programming Strategies Explicitly to Novice Programmers* PhD dissertation. University of Southern Queensland

de Raadt, M., Hamilton, M., Lister, R., Tutty, J., Baker, B., Box, I., Cutts, Q., Fincher, S., Haden, P., Hamer, J., Petre, M., Robins, A., Simon, Sutton, K., and Tolhurst, D. (2005) 'Approaches to learning in computer programming students and their effect on success'. *HERDSA* [online] 407-414. available from <https://www.cs.auckland.ac.nz/~j-hamer/HERDSA-05.pdf> [October 2014]

de Raadt, M., Watson, R., and Toleman, M. (2004) 'Introductory Programming: What's happening today and will there be any students to teach tomorrow?'. *Australian Computer Science Communications* 26, 277 -284

Dehnadi, S., and Bornat, R. (2006) 'The camel has two humps'. *Little IG 2006*. [online] available from <http://www.eis.mdx.ac.uk/research/PhDArea/saeed/paper1.pdf> [July 2015]

Dehnadi, S. (2009) *A cognitive study of learning to program in introductory programming courses*, PhD dissertation, [online] available from <http://eprints.mdx.ac.uk/6274/1/Dehnadi_A_Cognitive_Study_of_Learning.pdf> [March 2016]

Denny, P., and Luxton-Reily, A. (2008) 'The PeerWise System of Student Contributed Assessment Questions'. *Tenth Conference on Australasian Computing Education* 78

Denscombe, M. (2008) 'Communities of practice: a research paradigm for the mixed methods approach'. *Journal of Mixed Methods Research* 2 (3), 270-283

Department for Business Innovation and Skills (2011) *The plan for growth* [online] available from <https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/31584/2011budget_growth.pdf> [December 2015]

Department for Business innovation and Skills (2014) *Participation rates in Higher Education: academic years 2006/2007 – 2013/2014 (provisional)* [online] available from <https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/458034/HEIPR_PUBLICATION_2013-14.pdf> [November 2015]

Department for Business Innovation and Skills (2015) *Policy paper: 2010 to 2015 government policy: public understanding of science and engineering.* [online] available from <https://www.gov.uk/government/publications/2010-to-2015-government-policy-public-understanding-of-science-and-engineering/2010-to-2015-government-policy-public-understanding-of-science-and-engineering> [July 2015]

Department for Business Innovation and Skills (2016) *Success as a Knowledge Economy: Teaching Excellence, Social Mobility and Student Choice* [online] available from <https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/523546/bis-16-265-success-as-a-knowledge-economy-web.pdf> [April 2017]

Department for Education (2013a) *Computing programmes of study: key stages 1 and 2. National computing curriculum in England* [online] available from <https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/239033/PRIMARY_national_curriculum_-_Computing.pdf> [January, 2014]

Department for Education (2013b) *Computing programmes of study: key stages 3 and 4. National computing curriculum in England* [online] available from <http://www.computingatschool.org.uk/data/uploads/secondary_national_curriculum_-_computing.pdf> [January, 2014]

Department for Education (2014) *Nicky Morgan: closing the skills gap and our plan for education* [online] available from <https://www.gov.uk/government/speeches/nicky-morgan-closing-the-skills-gap-and-our-plan-for-education> [January 2019]

Department for Education (2015) *The link between absence and attainment at KS2 and KS4: 2012/13 academic year* [online] available from <https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/412638/The_link_between_absence_and_attainment_at_KS2_and_KS4.pdf> [June 2017]

Department for Education (2016) *Participation Rates In Higher Education: Academic Years 2006/2007 – 2014/2015 (Provisional)* [online] available from <https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/552886/HEIPR_PUBLICATION_2014-15.pdf> [April 2017]

Department for Education (2016b) *The link between absence and attainment at KS2 and KS4: 2013/14 academic year* [online] available from <https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/509679/The-link-between-absence-and-attainment-at-KS2-and-KS4-2013-to-2014-academic-year.pdf> [June 2017]

Doane, W. (2013) 'Computing Education Research. Part of: National Science Foundation (2013) Future Directions in Computing Education Summit white papers'. *Stanford Digital Repository*, [online] available from <http://purl.stanford.edu/mn485tg1952> [Jan 2018]

Dolnicar, S. (2005) 'Should we still lecture or just post examination questions on the web? The nature of the shift towards pragmatism in undergraduate lecture attendance'. *Quality in Higher Education* 11 (2), 103-115

Drummond, S. (2009) *Investigating the Impact of Entry Qualifications on Student Performance in Computing Programmes at Undergraduate Level*, Durham theses, Durham University, [online] available from <http://etheses.dur.ac.uk/154/> [October 2016]

Duncan, D. (2008) *Tips for Successful "Clicker" Use* [online] available from <http://casa.colorado.edu/~dduncan/clickers/Tips.htm> [June 2017]

Dunn, R. (1990) 'Understanding the Dunn and Dunn learning styles model and the need for individual diagnosis and prescription'. *Journal of Reading, Writing, & Learning Disabilities* 6 (3), 223-247

Dweck, C. (2008) *Mindset*. UK: Robinson

East, J. P., Thomas, S. R., Wallingford, E., Beck, W., and Drake, J. (1996) 'Pattern-based programming instruction'. In *1996 Annual Conference* 1.349.1-10

Eckerdal, A., and Berglund, A. (2005) What does it take to learn 'programming thinking'?' In *Proceedings of the First International Workshop on Computing Education Research* 135–143

Edexcel (2013) *UA035217 – Specification – Pearson Edexcel Level 3 GCE in Applied Information and Communication Technology (ICT) – Issue 5 – May 2013* [online] available from <http://qualifications.pearson.com/content/dam/pdf/A%20Level/Alied%20ICT/2013/Specification%20and%20sample%20assessments/UA035217_GCE_in_Alied_ICT_(Double_awd)_issue_5.pdf> [February 2017]

Edwards, E. G. (1982) *Higher Education for Everyone*. Russell Press Ltd. Nottingham

Edwards, H., Smith, B., and Webb, G. (2001) 'Introduction'. In H Edwards. B Smith and G Webb (eds) *Lecturing: Case Studies, Experience and Practice.* London: Kogan Page. 1-10

Eiriksdottir, E., and Catrambone, R. (2011) 'Procedural instructions, principles, and examples how to structure instructions for procedural tasks to enhance performance, learning, and transfer'. *Human Factors: The Journal of Human Factors and Ergonomics Society* 53 (6), 749-770

Elliot Tew, A., and Dorn, B. (2013) 'The Case for Validated Tools in Computer Science Education Research'. *Computer (IEEE)* 46 (9), 60-86

Elliot, A., McGregor, H., and Gable, S. (1999) 'Achievement goals, study strategies, and exam performance: A mediational analysis'. *Journal of Educational Psychology* 91 (3), 549-563

Engineering UK (2016) *Engineering UK 2016: The state of engineering* [online] available from <https://www.kinetic-plc.co.uk/media/EngineeringUK-Report-2016-Full-Report.pdf> [May 2017]

Engineering UK (2017) *Engineering UK 2017: The state of engineering* [online] available from <http://www.engineeringuk.com/media/1355/enguk-report-2017.pdf> [May 2017]

Entwistle, N. J., and Ramsden, P. (1983) *Understanding student learning*. Beckenham: Croom Helm

e-skills UK (2011) *Summary Technology Insights 2011*. Technology, London

Faiella, F., and Ricciardi, M. (2015) 'Gamification and learning: a review of issues and research'. *Journal of e-Learning and Knowledge Society* 11 (3)

Felder, R. M., and Solomon, B. A. (1997) *Index of Learning Styles Questionnaire* [online] available from <http://www.engr.ncsu.edu/learningstyles/ilsweb.html> [September 2017]

Felder, R. M., and Silverman, L. K. (1988) 'Learning and Teaching Styles in Engineering Education'. *Engineering Education* 78 (7), 674-681

Feldman, J. (2013) *Should university lectures be compulsory?* [online] available from <https://www.theguardian.com/education/mortarboard/2013/mar/14/should-university-lectures-be-compulsory> [June 2016]

Fincher, S., and Petre, M. (2004) *Computer Science Education Research*. Routledge: Falmer

Fincher, S. (1999) What are We Doing When We Teach Programming? In *Proceedings of 29th ASEE/IEEE Frontiers in Education Conference* 12a4, 1- 5. [online] available from <http://db.cs.duke.edu/courses/fall00/cps189s/readings/fincher-fie99.pdf> [October 2014]

Finlayson, R. (2018) *Richer data tells us more about students in higher education* Bristol: Office for Students. [online] available from <https://www.officeforstudents.org.uk/news-blog-and-events/news-and-blog/richer-data-tells-us-more-about-students-in-higher-education/> [Jan 2020]

Fisher, A., Margolis, J., and Miller, F., (1997) 'Undergraduate women in computer science: Experience, motivation and culture'. In *Proceedings of the 28th SIGCSE Technical Symposium on Computer Science Education* 106-110

Fisler, K., Krishamurthi, S., and Siegmund., J. (2016) 'Modernizing Plan-Composition Studies'. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE)* 211-216

Fleming, N.D. (1995) 'I'm different; not dumb: Modes of presentation (V.A.R.K.) in the tertiary classroom'. *Proceedings of the 1995 Annual Conference of the Higher Education and Research Development Society of Australasia (HERDSA)* 308-313

Ford, M., and Wenema, S. (2010) 'Assessing the Success of an Introductory Programming Course'. *Journal of Information Technology Education* 9, 133-145

Forte, A., and Guzdial, M. (2005) 'Motivation and Nonmajors in Computer Science: Identifying Discrete Audiences for Introductory Courses'. In *IEEE Transactions on Education* 48 (2), 248-253

Fotaris, P., Mastoras, T., Leinfellner, R., and Rosunally, Y. (2015) 'From Hiscore to High Marks: Empirical Study of Teaching Programming Through Gamification'. *European Conference on Games Based Learning (ECGBL 2015)* 186-194

Freeman, S., O'Connor, E., Parks, J. W., Cunningham, M., Hurley, D., Haak, D., Dirks, C., and Wenderoth, M. P. (2007) 'Prescribed active learning increases performance in introductory biology'. *CBE Life Sci Educ* [online] 6 (2), 132-139. available from <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1885904/> [September 2017]

Frey, K. (2017) *How Important is the Internet to Enterprises Today?* [online] available from <https://www.kentik.com/blog/todays-enterprises-rely-on-the-internet/> [Jan 2019]

Frey, T. (2011) *55 Jobs of the Future* [online] available from <http://www.futuristspeaker.com/business-trends/55-jobs-of-the-future/> [October 2017]

Fried, C. B. (2008) 'In-class laptop use and its effects on student learning'. *Computers & Education* 50, 906-914

Gazzaley, A., and Rosen, L. D. (2016) *The distracted mind: ancient brains in a high-tech world*. MIT Press: Cambridge MA

Geere, D. (2012) *Afterschool 'Code Clubs' planned to teach kids programming* [online] available from <https://www.wired.co.uk/article/code-club> [Jan 2019]

Gehringer, E. F., Deibel, K., et al. (2006) 'Panel: Cooperative Learning – Beyond Pair Programming and Team Projects'. *SIGCSE 2006 Technical Symposium on Computer Science Education* 458-459

Gibbs, D. C. (2000) 'The effect of a constructivist learning environment for field-dependent/independent students on achievement in introductory computer programming'. *ACM SIGCSE Bulletin* 32 (1), 207-211

Gibbs, G. (2015) 'Maximising Student Learning Gain'. In Fry, H., Ketteridge, S., Marshall, S. (2015) (eds.) *A handbook for teaching and learning in higher education.* (4th ed) Kogan Page: London. 193-208

Gicheva, N. and Petrie, K. (2018) *Vocation, Vocation, Vocation: The role of vocational routes into higher education: Social Market Foundation*. [online] available from <http://www.smf.co.uk/wp-content/uploads/2018/01/SMF-Vocation-Vocation-Vocation.pdf> [Jan 2020]

Gill, T., and Vidal Rodeiro, C. L. (2014) *Predictive validity of level 3 qualifications: Extended Project, Cambridge Pre-U, International Baccalaureate, BTEC Diploma. Cambridge Assessment Research Report. Cambridge, UK: Cambridge Assessment* [online] available from <http://www.cambridgeassessment.org.uk/Images/178062-predictive-validity-of-level-3-qualifications.pdf> [February 2017]

Glaser, B., and Strauss, A. L. (1967) *The discovery of grounded theory: strategies for qualitative research*. Chicago: Aldine

Glass, R., Ramesh, V., and Vessey, I. (2004) 'Computer science project work: Principle and pragmatics'. *Communications of the ACM* 47 (6), 89-94

Gokhale, A. A. (1995) 'Collaborative Learning Enhances Critical Thinking'. *Journal of Technology Education* 7 (1), 22-30

Goldman, R. (2014) 'Video representations and the perspectivity framework: epistemology, ethnography, evaluation, and ethics'. In *Video Research in the Learning Sciences,* eds. R. Goldman, R. Pea, B. Barron & S.J. Derry, Routledge: Oxford, 3-38

Gordon, N. A. (2016) *Issues in retention and attainment in Computer Science* [online] available from <https://www.heacademy.ac.uk/sites/default/files/retention_and_attainment_in_computer_science_0.pdf> [March 2016]

Gove, M. (2012) *Digital Literacy and the future of ICT in schools. Presentation at the BETT show, Department of Education* [online] available from

<http://education.gov.uk/inthenews/speeches/a00201868/michael-gove-speech-at-the-bett-show-2012> [September 2014]

Granneman, J. (2018) *Introverts Aren't Antisocial, They're Selectively Social* [online] available from <https://introvertdear.com/news/ways-that-introverts-socialize-differently-than-extroverts/> [April 2019]

Gray, S., St. Clair, C., James, R., and Mead, J. (2007) 'Suggestions for graduated exposure to programming concepts using fading Worked Examples'. In *Proceedings of the third international workshop on Computing education research (ICER '07)* 99-110

Gray, J., Boyle, T., and Smith, C. (1998) 'A constructivist learning environment implemented in Java'. In *Proceedings of ITiCSE '98* [online] 94-97. available from <https://www.idi.ntnu.no/~terjery/Gammelt/U4223/H06/Pensum/p94-gray.pdf> [June 2017]

Gregory, J., Trafton, G., and Reiser, J. (1993) *The contributions of studying examples and solving problems to skill acquisition* [online] available from <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.55.1180&rep=rep1&type=pdf> [July 2017]

Grove, J. (2016) *Should student attendance in classes be compulsory?* [online] available from <https://www.timeshighereducation.com/news/should-student-attendance-in-classes-be-compulsory> [October 2017]

Guo, P. (2014) *Python is Now the Most Popular Introductory Teaching Language at Top U.S. Universities* [online] available from <https://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-u-s-universities/fulltext> [June 2017]

Guzdial, M. (2016) *Learner-centred design of computing education: research on computing for everyone*. Morgan and Claypool: USA

Guzdial, M., Ericson, B., McKlin, T., and Engelman, S. (2012) 'A statewide survey on computing education pathways and influences: factors in broadening participation in computing'. In *ECER '12: Proceedings of the ninth annual international conference on International computing education research* 143-150

Guzdial, M., Ericson, B., Mcklin, T., and Engelman, S. (2014) 'Georgia Computes!: An intervention in a US state, with formal and informal education in a policy context'. *Transactions on Computing Education.* 14 (2), 1-29

Hagan, D., and Markham, S. (2000) 'Does it help to have some programming experience before beginning a computing degree program?'. *ITiCSE '00 Proceedings of the 5th annual SIGCSE/SIGCUE ITiCSE conference on Innovation and technology in computer science education* 25-28

Haggis, T. (2003) Constructing images of ourselves? A critical investigation into approaches to learning research'. *British Educational Research Journal* 29 (1), 89-104

Hake, R. R. (1998) 'Interactive-engagement vs. traditional methods: A six-thousand-student survey of mechanics test data for introductory physics courses'. *American Journal of Physics* 66 (1), 64-74

Halloun, I., and Hestenes, D. (1985) 'The initial knowledge state of college physics students'. *American Journal of Physics* 53 (11), 1043-1055

Hammer, R., Ronen, M., Sharon, A., Lankry, T., Huberman, Y., and Zamtsov, V. (2010) 'Mobile culture in college lectures: instructors' and students' perspectives'. *Interdisciplinary Journal of Knowledge and Learning Objects*, 6, 293-304. [online] available from <http://www.ijello.org/Volume6/IJELLOv6p293-304Hammer709.pdf> [April 2017]

Han, J., and Beheshti, M. (2010) 'Enhancement of computer science introductory courses with Mentored Pair Programming'. *Journal of Computing Sciences in Colleges* 25 (4)

Havergal, C. (2015) *9K-a-year students more 'career focused'* [online] available from <https://www.timeshighereducation.com/news/9k-year-students-more-career-focused> [December 2015]

Hegel, G. (1807) *Phänomenologie des Geistes*. Translated: *The Phenomenology of Spirit.* [translated in English by A. V. Miller in 1977] Oxford: Clarendon Press

Hembrooke, H., and Gay, G. (2003) 'The Laptop and the Lecture: The Effects of Multitasking in Learning Environments'. *Journal of Computing in Higher Education* 15 (1), 46-64

Hertz, M., and Ford, S. M. (2013) 'Investigating Factors of Student Learning in Introductory Courses'. In *Proceedings of the 44th ACM technical symposium on Computer Science Education (SIGCSE'13)* 165-170

Hertz, M. (2010) 'What do "CS1" and "CS2" mean?: investigating differences in the early courses'. In *Proceedings of the 41st ACM technical symposium on Computer science education* 199-203

High Fliers Research (2015) *The UK Graduate Careers Survey 2015* [online] available from <http://www.highfliers.co.uk/> [December 2015]

Higher Education Funding Council for England (2013) *Non-continuation rates at English HEIs: Trends for entrants 2005-06 to 2010-11* [online] available from <http://www.hefce.ac.uk/media/hefce/content/pubs/2013/201307/Non-continuation%20rates%20at%20English%20HEIs.pdf> [June 2015]

Higher Education Funding Council for England (2014) *Differences in Degree Outcomes: Key Findings* [online] available from <http://www.hefce.ac.uk/media/hefce/content/pubs/2014/201403/HEFCE2014_03.pdf> [December 2015]

Higher Education Funding Council for England (2015) *Young participation in higher education: A-levels and similar qualifications* [online] available from <http://www.hefce.ac.uk/pubs/year/2015/201503/> [November 2016]

Higher Education Funding Council for England (2016) *Destinations of Leavers from Higher Education survey* [online] available from <http://www.hefce.ac.uk/lt/dlhe/> [June 2017]

Hlubinka, M. (2015) *Teach your kids to code with one hour of Minecraft puzzles* [online] available from <http://makezine.com/2015/11/19/teach-your-kids-to-code-with-one-hour-minecraft-puzzles/> [December 2015]

Honey, P., and Mumford, A. (1992) *The manual of learning styles*. Maidenhead: Peter Honey Publications

Horton, C. (2018) *Recruitment crisis continues for UK IT industry* [online] available from <https://www.channelpro.co.uk/advice/10782/recruitment-crisis-continues-for-uk-it-industry> [Jan 2019]

House of Commons Education Select Committee (2018) *Value for money in higher education. Seventh Report of Session 2017-19* [online] available from <https://publications.parliament.uk/pa/cm201719/cmselect/cmeduc/343/343.pdf> [July 2019]

House of Commons Science and Technology Committee (2016) *Digital skills crisis: Second Report of Session 2016–17* [online] available from <https://www.publications.parliament.uk/pa/cm201617/cmselect/cmsctech/270/270.pdf> [May 2017]

House of Lords (2015) *Make or Break: The UK's digital future: Select Committee on Digital Skills, Report of Session 2014-15.* London: The Stationery Office

Hsieh, H. F., and Shannon, S. E. (2005) 'Three Approaches to Qualitative Content Analysis'. *Qualitative Health Research* 15 (9), 1277-1288

Hudak, M. A., and Anderson, D. E. (1990) 'Formal operations and learning style predict success in statistics and computer science courses'. *Teaching of Psychology* 17 (4), 231–234

Hundhausen, C.D., Farley, S.F., and Brown, J.L. (2009) 'Can direct manipulation lower the barriers to computer programming and promote transfer of training?: An experimental study'. *ACM Transactions in CHI* 16 (3), 1-13

Husserl, E. (1927) *Phenomenology.* Translated by C. V. Salmon. *The Encyclopaedia Britannica*, 14th ed. 17, 699-702

Huw, R. (1997) *The collision of two worlds* [online] available from <https://www.timeshighereducation.com/news/the-collision-of-two-worlds/104836.article?storyCode=104836&sectioncode=26> [Jan 2019]

Ihantola, P., Vihavainen, A., Ahadi, A., Butler, M., Borstler, J., Edwards, S.H., Isohanni, E., Korhonen, A., Peterson, A., Rivers, K., Rubio, M.A., Sheard, J., Skupas, B., Spacco, J., Szabo, C., and Toll, D. (2015) 'Educational data mining and learning analytics in programming: literature review and case studies'. *ITiCSE WGR '16* 41-63

Independent Commission on Fees (2015) *Independent Commission on Fees 2015 Final Report* [online] available from <https://www.suttontrust.com/wp-content/uploads/2015/07/ICOF-REPORT-2015.pdf> [May 2017]

Internet Advertising Bureau (2014) *Gaming Revolution* [online] available from <https://iabuk.net/system/tdf/research-docs/GamingRevolution-A0.pdf?file=1&type=node&id=22900> [September 2017]

Ipsos Mori and Lloyds Banking Group (2015) *Basic Digital Skills* [online] available from <http://s3-eu-west-1.amazonaws.com/digitalbirmingham/resources/Basic-Digital-Skills_UK-Report-2015_131015_FINAL.pdf> [April 2017]

Ismail, N. (2018) *Tech Nation 2018 report: UK tech expanding faster than the rest of the economy* [online] available from <https://www.information-age.com/tech-nation-2018-report-uk-tech-faster-economy-123471982/> [Jan 2019]

Jenkins, T. (2001a) The motivation of students of programming. PhD dissertation. University of Kent

Jenkins, T. (2001b) 'The Motivation of Students of Programming'. *ITICSE'01* [online] 53-56. available from <http://bioinfo.uib.es/~joe/semdoc/p53-jenkins.pdf> [June 2016]

Jenkins, T. (2001c) 'Teaching programming – a journey from teacher to motivator'. *In Proceedings of 2nd Annual conference of the LSTN Centre for Information and Computer Science* [online] 65-71. available from <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.584.6621&rep=rep1&type=pdf> [October 2014]

Jewitt, C. (2012) *An Introduction to Using Video for Research.* National Centre for Research Methods Working Paper, 03/12, Institute of Education: London

Jirotka, M., and Luff, P. (2006) 'Supporting requirements with video-based analysis'. *IEEE Software* 23 (3), 42-44

Jirotka, M. (1998) *Video-based elicitation: a manual*. Oxford University Computing Laboratory

Jirotka, M. and Wallen, L. (2000) 'Analysing the workplace and user requirements: challenges for the development of methods for requirements engineering'. In Luff, P., Hindmarsh, J., Heath, C. (eds.) *Workplace Studies: Recording Work Practice and Informing System Design.* Cambridge University Press

Johnson, D. (2013) *Schools haven't started thinking about coding curriculum* [online] available from <http://www.telegraph.co.uk/technology/news/10496515/Schools-havent-started-thinking-about-coding-curriculum.html> [May 2015]

Johnson, R. B., and Onwuegbuzie, A. J. (2004) 'Mixed methods research: A research paradigm whose time has come'. *Educational Researcher* 33 (7), 14-26

Joint Council for Qualifications (2013) *GCE Trends 2013* [online] available from <https://www.jcq.org.uk/examination-results/a-levels/2013/gce-trends-2013> [February 2018]

Joint Council for Qualifications (2014) *GCE Trends 2014* [online] available from <https://www.jcq.org.uk/examination-results/a-levels/2014/gce-trends-2014> [February 2018]

Joint Council for Qualifications (2015) *GCE Trends 2015* [online] available from <https://www.jcq.org.uk/examination-results/a-levels/2015/gce-trends-2015> [February 2018]

Joint Council for Qualifications (2016a) *GCE Trends 2016* [online] available from <https://www.jcq.org.uk/examination-results/a-levels/2016/gce-trends-2016> [February 2018]

Joint Council for Qualifications (2016b) *A, AS and AEA Results, Summer 2016* [online] available from <https://www.jcq.org.uk/examination-results/a-levels/2016/a-as-and-aea-results> [February 2018]

Jones, K. (1999) *A brief informal history of the Computer Laboratory* [online] available from <http://www.cl.cam.ac.uk/events/EDSAC99/history.html> [October 2014]

Joy, M., and Luck, M. (1998) *Plagiarism in Programming Assignments, Research Report 345* [online] available from <https://www.academia.edu/617369/Plagiarism_in_programming_assignments> [January 2016]

Kahne, J. (2008) 'Major New Study Shatters Stereotypes About Teens and Video Games'. John D. and Catherine T. MacArthur Foundation

Kalyuga, S., Ayres, P., Chandler, P., and Sweller, J. (2003) 'Expertise reversal effect'. *Educational Psychologist* 38, 23–31

Katz, S., Allbritton, D., Aronis, J., Wilson, C., and Soffa, M. L. (2006) 'Gender, achievement, and persistence in an undergraduate computer science program'. *ACM SIGMIS Database* 37 (4), 42-57

Kelly, S. (2017) *Reforming BTECs: Applied General qualifications as a route to higher education* Higher Education Policy Institute [online] available from <https://www.hepi.ac.uk/wp-content/uploads/2017/02/Hepi_Reforming-BTECs-Report-94-09_02_17-Web.pdf> [January 2020]

Kinnunen, P., and Malmi, L. (2006) 'Why Students Drop Out of a CS1 Course?' *In Proceedings of 2006 International workshop on Computing Education Research ICER '06.* 97-108

Kinnunen, P., Meisalo, V., and Malmi, L. (2010) 'Have we missed something? Identifying missing types of research in computing education'. *ICER '10* [online] 13-21. available from <http://ims.mii.lt/ims/konferenciju_medziaga/ICER'10/docs/p13.pdf> [October 2017]

Kirschner, P. A., and Karpinski, A. C. (2010) 'Facebook and academic performance'. *Computers in Human Behavior* 26, 1237–1245

Kirschner, P., Sweller, J., and Clark, R. (2006) 'Why minimal guidance during instruction does not work: an analysis of the failure of constructivist, discovery, problem-based, experimental, and inquiry-based teaching'. *Educational Psychologist* 41 (2), 75-86

Klahr, D., and Carver, S.M. (1988) 'Cognitive objectives in a LOGO debugging curriculum: Instruction, learning, and transfer'. *Cognitive Psychology* 20 (3), 362-404

Knight, J. K., and Wood, W. B. (2005) 'Teaching more by lecturing less'. *Cell Biology Education* 4 (4), 298-310

Kolb, D. A. (1984) *Experiential learning: Experience as the source of learning and development (Vol. 1)* Englewood Cliffs, NJ: Prentice-Hall

Kolb, D. A. (1985) *Learning style inventory (revised edition)* Boston: McBer and Company

Kolling, M., and Barnes, D. J. (2008) *Apprentice-Based Learning via Integrated Lectures and Assignments* [online] available from <https://www.academia.edu/2657417/Arentice-Based_Learning_Via_Integrated_Lectures_and_Assignments> [September 2015]

Kolling, M. (2003) *The curse of hello world*. Oslo, Norway: Invited lecture at Workshop on Learning and Teaching Object-Orientation - Scandinavian Perspectives

Koulouri, T., Lauria, S., and Macredie, R. D. (2014) 'Teaching Introductory Programming: a Quantitative Evaluation of Different Approaches'. *ACM Transactions on Computing Education* 14 (4), Article 26

Krathwohl, D. (1993) *Methods of educational and social science research: An integrated approach.* New York: Longman

Krause, L. (2000) *How we learn and why we don't*. Cincinnati, OH: Thomson Learning

Kraushaar, J., and Novak, D.C. (2006) 'Examining the Affects of Student Multitasking With Laptops During the Lecture'. *Journal of Information Systems Education* 21 (2), 241-251

Kriendorff, K. (2004) 2nd ed. *Content Analysis: An Introduction to Its Methodology*. Thousand Oaks, CA: Sage. 413

Kühn, S., Gleich, T., Lorenz, R. C, Lindenberger, U., and Gallinat, J. (2014) ‘Playing Super Mario induces structural brain plasticity: grey matter changes resulting from training with a commercial video game’. *Mol Psychiatry.* [online] 19, 265-271. available from <http://pubman.mpdl.mpg.de/pubman/item/escidoc:2096712:1/component/escidoc:2096711/SIK_Playing_2014.pdf> [August 2017]

Kurtz, B. L. (1980) ‘Investigating the relationship between the development of abstract reasoning and performance in an introductory programming class’. *ACM SIGCSE Bulletin* 12 (1), 110-117

Lave, J., and Wenger, E. (1991) *Situated Learning: Legitimate Peripheral Participation*. Cambridge: Cambridge University Press

Leafgran, F. A. (1989) ‘Health and wellness program’. In M.L. Upcraft & J.N. Gardner (Eds.), *The freshman year experience* (156-167) San Francisco: Jossey-Bass

Lee, M. J., and Ko, A. J. (2011) ‘Personifying programming tool feedback improves novice programmers’ learning’. In *Proceedings of the Seventh International Workshop on Computing Education Research (ICER’11)* 109-116

Leinikka, M., Vihavainen, A., Lukander, J., and Pakarinen, S. (2014) ‘Cognitive Flexibility and Programming Performance’. *IG, University of Sussex* 1-11

Lewis, S., McKay, J., and Lang, C. (2006) ‘The Next Wave of Gender Projects in IT Curriculum and Teaching at Universities’. *In Proceedings of the Eighth Australasian Computing Educational Conference (ACE2006)* [online] 52, 135 -142. available from <http://crpit.com/confpapers/CRPITV52Lewis.pdf> [November 2017]

Lewis, T. L., Chase, J. D., Perez-Quinones, M.A., and Rosson, M. B. (2005) ‘The effects of individual differences on CS2 course performance across universities’. *In SIGCSE '05: Proceedings of the 36th SIGCSE technical symposium on Computer science education* 426-430

Lipsett, A. (2009) *Huge increase in demand for postgraduate degree courses* [online] available from <https://www.theguardian.com/education/2009/feb/17/rise-alications-postgraduate-degrees> [January 2019]

Lishinski, A., Yadav, A., Enbody, R., and Good, J. (2016) ‘The influence of problem-solving abilities on students' performance on different assessment tasks in CS1’. *In SIGCSE 2016 - Proceedings of the 47th ACM Technical Symposium on Computing Science Education* 329-334

Lister, R. (2005a) ‘Computer Science Teachers as Amateurs, Students and Researchers’. In *Proceedings of the 5th Baltic Sea Conference on Computing Education Research, Koli Calling* 3-12

Lister, R. (2005b) ‘One Small Step Toward a Culture of Peer Review and Multi-Institutional Sharing of Educational Resources: A Multiple-Choice Exam for First Semester Programming Students’. In *Proceedings of Seventh Australasian Computing Education Conference (ACE 2005)* [online] 42, 155-164. available from <https://opus.lib.uts.edu.au/bitstream/10453/5869/3/2005003242.pdf> [July 2019]

Lister, R. (2016) ‘Toward a Developmental Epistemology of Computing Programming’. In *WiPSCE '16 Proceedings of the 11th Workshop in Primary and Secondary Computing Education* 11, 5-16

Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J. , Lindholm, M., McCartney, R., Moström, J. E., Sanders, K., Seälä, O., Simon, B., and Thomas, L. (2004) 'A multi-national study of reading and tracing skills in novice programmers'. *ACM SIGCSE Bulletin* 36 (4), 119-150

Lister, R., Cope, C., Bower, M., de Raadt, M., Mannila, L., Sheard, J., Traynor, D., Berglund, A., Pears, A., Carbone, A., Doyle, B., Kutay, C., Simon, Tutty, J., Box, I., Avram, C., Davey, B., Fitzgerald, S., Peltomaki, M., Sutton, K., and Venables, A. (2007) 'Differing Ways that Computing Academics Understand Teaching'. In *Proceedings of Ninth Australasian Computing Education Conference (ACE2007)* 66, 97-106

Livingstone, I., and Hope, A. (2011) *Next Gen. Transforming the UK into the world's leading talent hub for the video games and visual effects industries* [online] available from <https://www.nesta.org.uk/sites/default/files/next_gen_wv.pdf> [September 2014]

Livingstone, I. (2012) *A nation of digital illiterates* [radio recording] [online] available from <http://news.bbc.co.uk/today/hi/today/newsid_9675000/9675420.stm> [September 2014]

Livingstone, I., and Saeed, S. (2017) *Hacking the Curriculum: Creative Computing and the Power of Play.* John Catt Educational Ltd

Lloyd, J. (2013) *History [of Newcastle University's school of Computer Science]* [online] available from <http://www.ncl.ac.uk/computing/about/history/> [October 2014]

Lobel, B. (2016) *Three quarters of business owners rely on mobile or web applications* [online] available from <https://smallbusiness.co.uk/three-quarters-business-mobile-web-as-2534392/> [January 2019]

London Economics (2013) *The outcomes associated with the BTEC route of degree level acquisition: A report for Pearson* [online] available from <http://www.inteluni.ac.ke/docum/London%20Econimics%20report%20on%20BTEC%20HND%20route%2003-May-2013-Final%20Report.pdf> [February 2018]

Longi, K. (2016) *Exploring factors that affect performance on introductory programming courses* Master's Thesis. University of Helsinki

Luker, J. (2019) *Computing Assessment Board* [Discussion] [21st June 2019]

Lui, A. K., Khan, R., Poon, M., and Cheung, Y.H.Y. (2004) 'Saving weak programming students: applying constructivism in a first programming course'. *inroads – The SIGCSE Bulletin* 36 (2), 72-76

Lung, J., Aranda, J., Easterbrook, S., and Wilson, G. (2008) 'On the difficulty of Replicating Human Subjects Studies in Software Engineering'. In *ICSE '08. ACM/IEEE 30th International Conference on Software Engineering* 191-200

Luxton-Reily, A., Becker, B. A., Cao, Y., McDermott, R., Mirolo, C., Muhling, A., Peterson, A., Sanders, K., Simon, and Whalley, J. (2017) 'Developing Assessments to Determine Mastery of Programming Fundamentals'. *ITiCSE-WGR'17* 47 – 69

Lyon, L. A., and Denner, J. (2016) *Student Perspectives of Community College Pathways to Computer Science Bachelor's Degrees* [online] available from <https://services.google.com/fh/files/misc/student-perspectives-of-the-community-college-pathway-to-computer-science-report.pdf> [May 2017]

Ma, L., Ferguson, J., Roper, M., Ross, I., and Wood, M. (2009) 'Improving the Mental Models Held by Novice Programmers Using Cognitive Conflict and Jeliot Visualisations'. *ITiCSE'09* 166-170

MacGregor, J. (1988) 'Computer programming instruction: effects of collaboration and structured design mileposts'. *Journal of Research on Computing Education* 21, 155-164

Madden, M., and Zickuhr, K. (2011) *65% of online adults use social networking sites. Pew Research Centre* [online] available from <https://www.pewinternet.org/2011/08/26/65-of-online-adults-use-social-networking-sites/> [March 2016]

Maguire, P., Maguire, R., Hyland, P., and 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), 141:1-25

Malmi, L. (2010) 'Computing education and programming education research: challenges in field, research practices, methods and outcomes'. [presentation slides] Aalto University: School of Science and Technology, 10th February 2010

Malmi, L., Bednarik, R., Korhonen, A., Sheard, J., Helminen, J., Myller, N., Taherkhani, A., Simon, Kinnunen, P., and Sorva, J. (2014) 'Theoretical underpinnings of computing education research – what is the evidence?'. In *Proceedings of the tenth annual conference on International computing education research* 27-34

Maloney, J., Rush, N., Burd, L., Silverman, B., Kafai, Y., and Resnick, M. (2003) *Scratch: A Sneak Preview* [online] available from <https://www.researchgate.net/profile/Yasmin_Kafai/publication/4082665_Scratch_A_sneak_preview/links/55e4555f08aecb1a7cca5f10/Scratch-A-sneak-preview.pdf> [September 2017]

Margolis, J., Estrella, R., Goode, J., Home, J. J., and Nao, K. (2008) *Stuck in the Shallow End: Education, Race, and Computing*. MIT Press: Cambridge, MA

Margolis, J., and Fisher, A. (2002) *Unlocking the clubhouse: Women in Computing*. MIT Press. [online] available from <https://pdfs.semanticscholar.org/8e12/42e37d71d9e592201a8ed8bc468caf07fef8.pdf> [September 2017]

Margulieux, L. E., Guzdial, M., and Catrambone, R. (2012) 'Subgoal-labeled instructional material improves performance and transfer in learning to develop mobile applications.' *In Proceedings of the Ninth Annual International Conference on International Computing Education Research* 71-78

Margulieux, L. E., and Catrambone, R. (2014) 'Improving problem-solving performance in computer-based learning environments through subgoal labels'. *Proceedings of the first ACM conference on Learning@ scale conference* 149-150

Margulieux, L. E., and Catrambone, R. (2016) 'Using Subgoal Learning and Self-Explanation to Improve Programming Education'. In Papafragou, A., Grodner, D., Mirman, D., and Trueswell, J.C. (Eds.), In *Proceedings of the 38th Annual Conference of the Cognitive Science Society (2009-2014) Austin, TX: Cognitive Science Society*, 2016. [online] available from <http://scholarworks.gsu.edu/cgi/viewcontent.cgi?article=1006&context=ltd_facpub> [June 2017]

Margulieux, L. E., Catrambone, R., and Guzdial, M. (2013) 'Subgoal labelled Worked Examples improve K-12 teacher performance in computer programming training'. In Knauff, M., Pauen, M., Sebanz, N., and Wachsmuth, I., eds, *Proceedings of the 35th Annual Conference of the Cognitive Science Society* 71-78

Marshall, R. (2013) *UK faces severe shortage of computer science teachers* [online] available from <https://www.v3.co.uk/v3-uk/news/2239958/exclusive-uk-faces-severe-shortfall-of-computer-science-teachers> [May 2015]

Martinez, P. (2001) *Improving student retention and achievement. What do we know and what do we need to find out?* London, Learning and Skills Development Agency. [online] available from <https://core.ac.uk/download/pdf/4153982.pdf> [September 2017]

Marton, F., and Booth, S. (1997) *Learning and awareness.* New Jersey: Lawrence Erlbaum Associates

Marton, F., and Saljo, R. (1976) 'On qualitative differences in learning — 2: outcome as a function of the learner's conception of the task'. *British Journal of Educational Psychology* 46, 115-127

Mather, R. (2015) 'Multivariate Gradient Analysis for Evaluating and Visualizing a Learning Platform for Computer Programming'. *IAFOR Journal of Education* 3 (1), 17-30

Mather, R. (2004) *Report to episitec games: evaluation of ceebot as a technology for supporting the learning of computer programming.* Masters assignment. Oxford University. [unpublished]

Mather, R. (2014) 'Multivariate Gradient Analysis for Evaluating and Visualizing a Learning System Platform for Computer Programming'. *The European Conference on Technology in the Classroom 2014: Official Conference Proceedings* [online] 111-126. available from <http://iafor.org/archives/proceedings/ECTC/ECTC2014_proceedings.pdf> [November 2015]

Mather, R., Day, N., Jones, R., Lusuardi, C., Maher, K., Dexter, B. (2015) 'Canonical explorations of 'TEL' Environments for Computer Programming'. *The European Conference on Technology in the Classroom 2015: Official Conference Proceedings* [online] 265 – 282. available from <http://iafor.org/archives/proceedings/ECTC/ECTC2015_proceedings.pdf> [November 2015]

Mathieson, S. (2015) 'Student Learning' In Fry, H., Ketteridge, S., Marshall, S. (2015) (eds.) *A handbook for teaching and learning in higher education.* Kogan Page: London. 63-79

Mayer, R. E., Dyck, J. L., and Vilberg, W. (1986) 'Learning to program and learning to think: what's the connection?'. *Communications of the ACM* 29 (7), 605-610

Mazur, E. (1997) *Peer Instruction: A User's Manual*. Pearson: Harlow, UK

McConnell C. R., and Lamphear, C. (1969) 'Teaching principles of economics without lectures'. *Journal of Economic Education* 1 (4), 20-32

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y.B.D, Laxter, C., Thomas, L., Utting, I., and Wilusz, T. (2001) 'A multi-national, multi-institutional study of assessment of programming skills of first-year CS students'. *ACM SIGCSE Bulletin* [online] 33 (4), 125-140. available from <https://kar.kent.ac.uk/13514/4/a_multinational_ulti-institutional_mccracken.pdf> [June 2015]

McDermott, L. C. (1991) 'Millikan Lecture 1990: What we teach and what is learned - Closing the gap'. *American Journal of Physics* 59, 301-315

McDowell, C., Werner, L., and Fernald, J. (2002) 'The effects of pair-programming on performance in an introductory programming course'. In *ACM SIGCSE Bulletin* 34, 38-42

McDowell, C., Werner, L., and Fernald, J. (2006) 'Pair Programming improves student retention, confidence, and program quality'. *Communications of the ACM* [online] 49 (8), 90-95. available from <https://cloudfront.escholarship.org/dist/prd/content/qt1s49s13f/qt1s49s13f.pdf> [September 2017]

McIver, L., and Conway, D. (1996) ‘Seven deadly sins of introductory programming language design’. *In Proc. Software Engineering: Education and Practice* 309-316

McKetta, I. (2017) *The World’s Internet Speeds Increased More than 30% in 2017. Are You Keeping Up?* [online] available from <https://www.speedtest.net/insights/blog/global-speed-2017/> [January 2019]

McKinney, D., and Denton, L. F. (2006) ‘Developing Collaborative Skills Early in the CS Curriculum in a Laboratory Environment’. *SIGCSE 2006 Technical Symposium on Computer Science Education* 38, 138-142

McQuiggan S. W., Lee S., and Lester J. C. (2007) ‘Early Prediction of Student Frustration’. In Paiva A.C.R., Prada R., Picard R.W. (eds) Affective Computing and Intelligent Interaction. ACII 2007. Lecture Notes in Computer Science, 4738. 698 – 709. Springer, Berlin, Heidelberg. [online] available from <https://www.intellimedia.ncsu.edu/wp-content/uploads/frustration-acii-2007.pdf> [March 2017]

Meyer, B. (2003) *The Outside-In method of teaching introductory programming* [online] available from <http://se.ethz.ch/~meyer/publications/teaching/teaching-ispj.pdf> [May 2015]

Milkova, E., and Petranek, K. (2015) ’Improving programming courses using aptitude testing and learning styles’. *Recent advances in Computer Science* [online] 211-214. available from <http://www.inase.org/library/2015/zakynthos/bypaper/COMPUTERS/COMPUTERS-33.pdf> [September 2017]

Miller, G. (1956) ’The magical number seven, plus or minus two: Some limits on our capacity for processing information’. *The psychological review* 63, 81-97

MillionPlus (2018) *Forgotten Learners: building a system that works for mature students* [online] available from <http://www.millionplus.ac.uk/documents/Forgotten_learners_building_a_system_that_works_for_mature_students.pdf> [March 2019]

Ministry of Education (2004) *Historical Overview of Education in Sri Lanka* [online] available from <https://web.archive.org/web/20070716215924/http://www.moe.gov.lk/modules.php?name=Content&pa=showpage&pid=2#s1_4> [October 2018]

Molesworth, M., Scullion, R., and Nixon, E. (2011) *The Marketisation of Higher Education*. London: Routledge

Moreno, R., Reisslein, M., and Delgoda, G. (2006) ‘Toward a fundamental understanding of worked example instruction: Impact of means-ends practice, backward/forward fading, and adaptivity’. *Frontiers in Education, Annual* 0, 5-10

Morrison, B. B., Margulieux, L. E., Ericson, B., and Guzdial, M. (2016) ‘Subgoals Help Students Solve Parsons Problems’. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE 2016)* 42-47

Morrison, B. B., Margulieux, L. E., and Guzdial, M. (2015) ‘Subgoals, context, and Worked Examples in learning computer problem solving’. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research (ICER’15)* 21-29

Murphy, E., Crick, T., and Davenport, J. (2016) ‘An Analysis of Introductory University Programming Courses in the UK’. Bath: University of Bath

Myers, I. B., and McCaulley, M. H. (1985) *Manual: a guide to the development and use of the Myers-Briggs Type Indicator.* Palo Alto, USA: Consulting Psychologists Press

National Committee of Inquiry into Higher Education (1997) *Higher Education in the Learning Society* (the Dearing Report) London: HMSO

National Science Foundation and US Department of Education (2013) *Common Guidelines for Educational Research* [online] available from <http://www.nsf.gov/pubs/2013/nsf13126/nsf13126.pdf> [October 2015]

Naughton, J. (2012) *Why all our kids should be taught to code* [online] available from <http://www.theguardian.com/education/2012/mar/31/why-kids-should-be-taught-code> [December 2015]

Navarro-Lopez, E. (2014) *67 years of computing history at Manchester* [online] available from <http://www.cs.manchester.ac.uk/about-us/history/> [October 2014]

Nawahdah, M., and Taji, D. (2015) 'Work in Progress: Investigating the Effects of Pair-Programming on Students' Behaviour in an advanced Computer Programming Course'. *2015 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)* 157-160

NCET (1993) *A Survey of IT in Schools National Council for Educational Technology*. London

Nesta (2013) *Next Gen. Next Steps* [online] available from <https://www.nesta.org.uk/sites/default/files/next-gen-next-steps.pdf> [December 2015]

NESTA (2015) *Young Digital Makers Surveying attitudes and opportunities for digital creativity across the UK* [online] available from <https://media.nesta.org.uk/documents/youngdigmakers.pdf> [February 2019]

New Labour Party (2001) *Ambitions for Britain: labour's manifesto 2001* [online] available from <http://www.politicsresources.net/area/uk/e01/man/lab/ENG1.pdf> [November 2015]

Ni, L. (2009) 'What makes CS teachers change? Factors influencing CS teachers' adoption of curriculum innovations'. *SIGCSE Bull.* 41 (1), 544-548

Nicholls, J. G. (1984) 'Achievement motivation: Conceptions of ability, subjective experience, task choice, and performance'. *Psychological Review* [online] 91, 328–346. available from <https://pdfs.semanticscholar.org/3302/5ae403ab6a3c3a44fc5b69d539cb26133aa3.pdf> [December 2017]

Nikula, U., Gotel, O., and Kasurinen, J. (2011) 'A motivation guided holistic rehabilitation of the first programming course'. *ACM Trans. Comput. Educ.* [online] 11 (4), 24: 1-30. available from <https://www.viope.com/assets/uploads/Nikula2011-AMotivationGuidedHolisticRehabilitationOfTheFirstProgrammingCourse-1.pdf> [December 2017]

No Author. (undated) *Education in Sri Lanka* [online] available from <https://en.wikipedia.org/wiki/Education_in_Sri_Lanka> [September 2017]

Norvig, P. (2001) *Teach Yourself Programming in Ten Years* [online] available from <http://www.norvig.com/21-days.html> [June 2017]

Nosek, J. T. (1998) 'The Case for Collaborative Programming'. *Communications of the ACM* 41 (3), 105-108

OCR (2009) *GCE Computing: Mark Schemes for the Units June 2009* [online] available from <http://study-computing.co.uk/a-level/ocr/downloads/exam-papers/old-specification/june-2009/57674-mark-scheme-june.pdf> [February 2017]

OCR (2013) *AS/A Level GCE ICT: Version 5 - September 2013 Specification* [online] available from <http://www.ocr.org.uk/Images/81815-specification.pdf> [February 2017]

OCR (2015) *AS/A Level GCE Computing: Version 4 – September 2015* [online] available from <http://www.ocr.org.uk/Images/70162-specification.pdf> [February 2017]

OCR (2018) *A-level Computer Science Specification* [online] available from <https://www.ocr.org.uk/Images/170844-specification-accredited-a-level-gce-computer-science-h446.pdf> [June 2019]

Ofcom (2014) *Adults media use and attitudes report* [online] available from <http://stakeholders.ofcom.org.uk/binaries/research/media-literacy/adults-2014/2014_Adults_report.pdf> [May 2015]

Ofqual (2018) *An exploration of grade inflation in 'older style' level 3 BTEC Nationals* [online] available from <https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/763761/BTEC_grade_inflation_report_-_FINAL64592.pdf> [July 2019]

OFFA and OU (2017) *Understanding the impact of outreach activity for mature learners with low or no prior qualifications* [online] available from <https://www.offa.org.uk/egp/improving-evaluation-outreach/outreach-mature-learners/> [March 2016]

OFFA (2018a) *Topic briefing: Black and minority ethnic (BME) students* [online] available from <https://www.officeforstudents.org.uk/media/145556db-8183-40b8-b7af-741bf2b55d79/topic-briefing_bme-students.pdf> [March 2019]

OFFA (2018b) *Topic briefing: mature and part-time students* [online] available from <https://webarchive.nationalarchives.gov.uk/20180511111626/https://www.offa.org.uk/universities-and-colleges/guidance/topic-briefings/offa-topic-briefing-mature-learners/> [March 2019]

Office for Fair Access (2015a) *OFFA Strategic plan 2015-2020* [online] available from <https://www.offa.org.uk/wp-content/uploads/2015/03/OFFA-Strategic-Plan-2015-2020.pdf> [December 2015]

Onwuegbuzie, A. J., and Leech, N. L. (2005) 'On becoming a pragmatist researcher: The importance of combining quantitative and qualitative research methodologies'. *International Journal of Social Research Methodology: Theory & Practice* 8, 375-387

Organisation for Economic Co-operation and Development (2005) *Formative assessment: Improving student learning in secondary classrooms*. Centre for Educational Research and Innovation

Oxford Institute for the Advancement of University Learning (n.d) *Paper 2 - student approaches to learning* [online] available from <https://www.learning.ox.ac.uk/media/global/wwwadminoxacuk/localsites/oxfordlearninginstitute/documents/suortresources/lecturersteachingstaff/resources/resources/Student_Aroaches_to_Learning.pdf> [October 2014]

Palumbo, D. (1990) 'Programming Language / Problem-Solving Research: A Review of Relevant Issues'. *Review of Educational Research* 60 (1), 65

Parsons, D., Wood, K., and Haden, P. (2015) 'What Are We Doing When We Assess Programming?' In *Proceedings Of The 17th Australasian Computing Education Conference (ACE 2015)* 119-127

Pashler, H., McDaniel, M., Rohrer, D., and Bjork, R. (2009) 'Learning styles: concepts and evidence'. *Psychological science in the public interest* 9 (3), 105-119

Pears, A., East, P., McCartney, R., Ratcliffe, M.B., Stamouli, I., Berglund, A., Kinnunen, P., Mostrom, J.E., Schulte, C., Eckerdal, A., Malmi, L., Murphy, L., Simon, B., and Thomas, L. (2008) 'What's the problem? Teachers' experience of student learning successes and failures'. *Proceedings of the Seventh Baltic Sea Conference on Computing Education Research* [online] 88, 207-211. available from <http://crpit.com/confpapers/CRPITV88Pears.pdf> [November 2015]

Pears, A., Seidman, S., Eney, C., Kinnunen, P., and Malmi, L. (2005) 'Constructing a core literature for computing education research'. *ACM SIGCSE* 37 (4), 152-161

Pearson (2010b) *BTEC Level 3 National Creative Media Production Student Book* [online] available from <http://www.pearsonschoolsandfecolleges.co.uk/FEAndVocational/CreativeandMedia/BTECLevel%203/Level3BTECNationalCreativeandMediaProduction/Samples/Takeapeek/9781846906725_takeapeek.pdf> [December 2017]

Pearson (2010c) *Creative Media Production* [online] available from <https://qualifications.pearson.com/content/dam/pdf/BTEC-Nationals/Creative-Media-Production/2010/Specification/9781446934593_BTEC_90c_L3_CMediaProd_Iss3.pdf> [December 2017]

Pearson (2013) *ICT Specification – Pearson Edexcel Level 3 GCE in Alied Information and Communication Technology (ICT) – Issue 5 – May 2013* [online] available from <https://qualifications.pearson.com/content/dam/pdf/A%20Level/Alied%20ICT/2013/Specification%20and%20sample%20assessments/UA035217_GCE_in_Alied_ICT_(Double_awd)_issue_5.pdf> [December 2017]

Pearson (2015) *Art and Design Interactive Media Specification* [online] available from <https://qualifications.pearson.com/content/dam/pdf/BTEC-Higher-Nationals/Interactive-Media/2010/Specification/9781446926000_BTEC_HNS_L45_IntMed_Issue_5.pdf> [April 2018]

Pearson (2016) *Pearson BTEC Level 4 Higher National Certificate in Interactive Media & Pearson BTEC Level 5 Higher National Diploma in Interactive Media - Specification, Issue 6* [online] available from <https://qualifications.pearson.com/content/dam/pdf/BTEC-Higher-Nationals/Interactive-Media/2010/Specification/BTEC_Higher-Nationals_Interaticve-Media_specification-Issue_6.pdf> [April 2018]

Pearson (2016b) *Pearson BTEC Level 3 National Extended Diploma in Computing – Specification, Issue 5* [online] available from <https://qualifications.pearson.com/content/dam/pdf/BTEC-Nationals/computing/2016/specification-and-sample-assessments/9781446937907_BTEC_Nat_ExtCert_COMP_SPEC_Iss2C.pdf> [June 2019]

Pearson (2016c) *Pearson BTEC Level 3 National Extended Diploma in Information Technology – Specification, Issue 4* [online] available from <https://qualifications.pearson.com/content/dam/pdf/BTEC-Nationals/Information-Technology/2016/specification-and-sample-assessments/9781446938041_BTEC_NAT_Cert_IT_Spec_Iss2CUpd.pdf> [June 2019]

Pedroni, M., and Meyer, B. (2006) 'The Inverted Curriculum in Practice'. In *Proceedings of SIGCSE 2006* [online] 481-485. available from <http://se.ethz.ch/~meyer/publications/teaching/sigcse2006.pdf> [September 2017]

Perrotta, C., Featherstone, G., Aston, H., and Houghton, E. (2013) *Game-based Learning: Latest Evidence and Future Directions* (NFER Research Programme: Innovation in Education). Slough: NFER. [online] available from <http://www.nodo-observa.es/sites/default/files/GAME01.pdf> [May 2017]

Petersen, C. G., and Howe, T. G. (1979) 'Predicting Academic Success in Introduction to Computers'. *AEDS Journal* 12 (4), 182-191

Peterson, L. R., and Peterson, M. J. (1959) 'Short-term retention of individual verbal items'. *Journal of experimental psychology* 58 (3), 193-198

Phillips, R. and Gilding, T. (2003) 'Approaches to Evaluating the Effect of ICT on Student Learning'. *ALT Starter Guide 8* [online] available from <https://www.alt.ac.uk/sites/default/files/assets_editor_uploads/documents/eln015.pdf> [November 2015]

Piaget, J. (1950) *The Psychology of Intelligence*. London: Routledge and Kegan Paul

Piaget, J. (1952) *The Origins of Intelligence in Children*. New York: International University Press

Pioro, B. T. (2006) 'Introductory computer programming: gender, major, discrete mathematics and calculus'. *Journal of Comput. Small Coll.* 21 (5), 123-129

PISA (2003) PISA Problem Solving Items and Scoring Guides

Plociniczak, H. (2015) *Decrypting Local Type Inference* [online] PhD Dissertation. available from <https://infoscience.epfl.ch/record/214757/files/EPFL_TH6741.pdf?version=1> [July 2019]

Plonka, L., Sharp, H., and van der Linden, J. (2011) *Knowledge Transfer in Pair Programming: An In-depth Analysis* [online] available from <http://oro.open.ac.uk/41032/8/41032ORO.pdf> [February 2016]

Pocius, K. E. (1991) 'Personality factors in human-computer interaction: A review of the literature'. *Computers in Human Behavior* 7 (3), 103-135

Porter, L., and Simon, B. (2013) 'Retaining nearly one-third more majors with a trio of instructional best practices in CS1'. In *Proceedings of the 44th Special Interest Group on Computer Science Education Technical Symposium* 165-170

Porter, L., Bailey-Lee, C., and Simon, B. (2013a) 'Halving Fail Rates using Peer Instruction: a study of four computer science courses'. In *Proceedings of the 44th ACM technical symposium on Computer science education* 177-182

Porter, L., Bailey-Lee, C., Simon, B., Cutts, Q., and Zingaro, D. (2011) 'Experience report: a multi-classroom report on the value of Peer Instruction'. In *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education* 138-142

Porter, L., Bouvier, D., Cutts, Q., Grissom, S., Lee, C., McCartney, R., Zingaro, D., and Simon, B. (2016) 'A Multi-institutional Study of Peer Instruction in Introductory Computing'. *SIGCSE '16* 358-363

Porter, L., Guzdial, M., McDowell, C., and Simon, B. (2013b) 'Success in introductory programming: What works?' *Communications of the ACM* 56 (8), 34-36

Preston, D. (2006) 'Using collaborative learning research to enhance Pair Programming pedagogy'. *ACM SIGITE Newsletter* 3 (1), 16-21

Putwain, D. (2008) *Examination stress and test anxiety* [online] available from <https://thepsychologist.bps.org.uk/volume-21/edition-12/examination-stress-and-test-anxiety#> [April 2018]

R Core Team (2013) *R: A language and environment for statistical computing.* R Foundation for Statistical Computing. [online] available from <http://www.R-project.org/> [June 2019]

Radenski, A. (2007) ‘Digital Abductive Learning in Early Programming Courses’. *SIGCSE 2007 - Learning Solutions for the First Year Session* 14-18

Radermacher, A., Walia, G., and Rummelt, R. (2012) ‘Assigning Student Programming Pairs Based on their Mental Model Consistency: An Initial Investigation’. *SIGCSE’12* 325-330

Ramalingam, V., and Wiedenbeck, S. (1998) ‘Development and Validation of Scores on a Computer Programming Self-efficacy Scale and Group Analysis of Novice Programmer Self-efficacy’. *Journal of Educational Computing Research* 19 (4), 367-381

Randolph, J. J. (2008) *Multidisciplinary methods in educational technology research and development.* Published by HAMK Press/Justus Randolph [online] available from <http://justusrandolph.net/articles/multidisciplinary_methods.pdf> [December 2017]

Randolph, J. J., Julnes, G., Sutinen, E., and Lehman, S. (2007) ‘Findings from “A Methodological Review of the Computer Science Education Research: 2000-2005”’. *SIGCSE Bulletin* 39 (4), 130-157

Reams, P., and Twale, D. (2008) ‘The promise of mixed methods: discovering conflicting realities in the data’. *International Journal of Research and Method in Education* 31 (2), 133-142

Reidy, T. (2015) *Will taking a BTEC help or hinder your university application?* [online] available from <https://www.theguardian.com/education/2015/jul/21/will-taking-a-btec-help-or-hinder-your-university-alication> [December 2016]

Reis, R. (1998) ‘A Learning-Centered Approach to Course and Curriculum Design’ in *Designing and Assessing Courses and Curricula: A Practical Guide* by Robert M. Diamond. Jossey Bass: San Francisco [online] 6-8. available from <https://tomprof.stanford.edu/posting/442> [January 2016]

Richardson, M., Bond, R., and Abraham, C. (2012) Psychological correlates of university students’ academic performance: a review and meta-analysis. *Psychological Bulletin* [online] 138 (2), 353-387. available from <http://emilkirkegaard.dk/en/wp-content/uploads/Psychological-correlates-of-university-students-academic-performance.pdf> [July 2014]

Rideout, V. J., Foehr, U. G., and Roberts, D. F. (2010) *GENERATION M2 Media in the Lives of 8- to 18-Year-Olds A Kaiser Family Foundation Report* [online] available from <http://www.kff.org/entmedia/upload/8010.pdf> [July 2017]

Robbins, L. C. (1963) *Committee on Higher Education report* [online] available from <http://www.educationengland.org.uk/documents/robbins/robbins1963.html> [May 2017]

Roberts, E. (2006) ‘An Interactive Tutorial System for Java’. *SIGCSE 2006 Technical Symposium on Computer Science Education* 334-348

Roberts, M. R. H., McGill, T., and Koi, T. (2011) ‘What students are telling us about why they left their ICT course’. *Innovation in Teaching and Learning in Information and Computer Sciences* 10 (3), 68-83

Robins, A. (2010) ‘Learning edge momentum: a new account of outcomes in CS1’. *Computer Science Education* [online] 20 (1), 37-71. available from <http://reflect.otago.ac.nz/staffpriv/anthony/publications/pdfs/RobinsLEM.pdf> [October 2014]

Robins, A. (2015) ‘The ongoing challenges of computer science education research’. *Computer Science Education* 25 (2), 115-119

Robins, A., Rountree, J., and Rountree, N. (2003) 'Learning and Teaching Programming: A Review and Discussion'. *Computer Science Education* 13 (2), 137-172

Robson, C. (2002) *Real World Research: A Resource for Social Scientists and Practitioner-researchers*. 2nd ed. Oxford, United Kingdom

Roddan, M. (2002) *The determinants of student failure and attrition in first year computing science*. Final-year undergraduate project. [online] available from <http://www.psy.gla.ac.uk/~steve/localed/roddanpsy.pdf> [January 2016]

Rodriguez, F. J., Price, K. M., and Boyer, K. E. (2017) 'Exploring the Pair Programming Process: Characteristics of Effective Collaboration'. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* 507-512

Rogerson, C., and Scott, E. (2010) 'The fear factor: how it affects students learning to program in a tertiary environment'. *Information Technology Education* [online] 9 (1), 147-171. available from <http://jite.org/documents/Vol9/JITEv9p147-171Rogerson803.pdf> [May 2017]

Rojas, R., Ganjineh, T., and Vogel, P. (2011) *Brandenburg Gate Mission Accomplished: Autonomous Car Navigates the Streets of Berlin* [online] available from <https://www.fu-berlin.de/en/presse/informationen/fup/2011/fup_11_291/> [January 2019]

Rokicka, M. (2014) *The impact of students' part-time work on educational outcomes* [online] available from <https://www.iser.essex.ac.uk/research/publications/working-papers/iser/2014-42.pdf> [October 2017]

Rosen, L. D., Linn., A. F., Carrier, L. M, and Cheever, N. A. (2011) 'An Empirical Examination of the Educational Impact of Text Message-induced Task Switching in the Classroom: Educational Implications and Strategies to Enhance Learning'. *Psicologia Educativa* 17 (2), 163-177

Rountree, N., Rountree, J., and Robins, A. (2002) 'Predictors of success and failure in a CS1 course'. *inroads SIGCSE Bulletin* 34 (2), 121-124

Russell Group (2017) *Informed Choices: A Russell Group guide to making decisions about post-16 education Cambridge: The Russell Group of Universities* [online] available from <https://russellgroup.ac.uk/media/5686/informed-choices-2018-1-6th-edition-final.pdf> [Jan 2020]

Ryan, D. (2016) *How Sensor Technologies are Transforming Agriculture* [online] available from <http://connectedbusiness.uscellular.com/how-sensor-technologies-are-transforming-agriculture/> [May 2017]

Sadler-Smith, E., Spicer, D. P. and Tsang, F. (2000) 'Validity of the Cognitive Style Index: replication and extension'. *British Journal of Management* 11: 175-181

SAITM (undated) *BTEC HND MEDIA* [online] available from <http://www.saitm.edu.lk/fac_of_media/btec_hnd_media.htm> [April 2018]

Sana, F., Weston, T., and Cepeda, N. J. (2013) 'Laptop multitasking hinders classroom learning for both users and nearby peers'. *Computers and Education* 62, 24-31

Saunders, D. B. (2014) 'They do not buy it: exploring the extent to which entering first-year students view themselves as customers'. *Journal of Marketing for Higher Education* 25, 5-28

Savage, M. (2018) *Shortfall in teacher numbers hits 30,000* [online] available from <https://www.theguardian.com/education/2018/feb/04/30000-teacher-shortfall--secondary-schools-further-education> [July 2019]

Schaeffer, L. M., Margulieux, L. E., and Catrambone, R. (2016) 'Interaction of instructional materials order and subgoal labels on learning in programming'. In Papafragou, A., Grodner, D., Mirman, D. and Trueswell, J. C. (Eds.), Proceedings of the 38th Annual Conference of the Cognitive Science Society (271-276) Austin, TX: Cognitive Science Society, 2016. [online] available from <http://scholarworks.gsu.edu/cgi/viewcontent.cgi?article=1007&context=ltd_facpub> [June 2017]

Schiller, B. (2015) *How Self-Driving Cars Will Change The Economy And Society* [online] available from <https://www.fastcompany.com/3043305/how-self-driving-cars-will-change-the-economy-and-society> [January 2019]

Schwab, K. (2016) *The Fourth Industrial Revolution*. World Economic Forum: Switzerland

Sentence, S., and Csizmadia, A. (2016) 'Computing in the curriculum: Challenges and strategies from a teacher's perspective'. In *Education and Information Technologies* 1-27

Sfetsos, P., Stamelos, I., Angelis, L., and Deligiannis, I. (2009) 'An experimental investigation of personality types impact on pair effectiveness in Pair Programming'. *Empirical Software Engineering* 14 (2), 187-226

Shadbolt, N. (2016) *Shadbolt Review of Computer Sciences Degree Accreditation and Graduate Employability* [online] available from <https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/518575/ind-16-5-shadbolt-review-computer-science-graduate-employability.pdf> [February 2017]

Sheard, J., Simon, Hamilton, M., and Lonnberg, J. (2009) 'Analysis of Research into the Teaching and Learning of Programming'. *ICER '09* 93-104

Shell, D. F., Soh, L. K., Flanigan, A. E., and Peteranetz, M. S. (2016). 'Students' Initial Course Motivation and Their Achievement and Retention in College CS1 Courses'. *SIGCSE '16* 639-644

Shih, T. H., and Fan, X. (2009) 'Comparing response rates in e-mail and paper surveys: a meta-analysis'. *Educational Research Review* 4 (1), 26-40

Simon, B., and Cutts, Q. (2012a) 'Peer Instruction: a teaching method to foster deep understanding'. In *Communications of the ACM* [online] 55 (2), 27-29. available from <https://cacm.acm.org/magazines/2012/2/145404-peer-instruction/fulltext> [May 2015]

Simon, B., and Cutts, Q. (2012b) *How to Implement a Peer Instruction-designed CS Principles Course* [online] available from <http://www.peerinstruction4cs.org/wp-content/uploads/2012/07/Simon-formattedInroadsPre-print.pdf> [May 2015]

Simon, B., Kohanfars, M., Lee, J., Tamayo, K., and Cutts, Q. (2010) 'Experience report: Peer Instruction in introductory computing'. *Proc. of 41st SIGCSE* 341-345

Simon, B., Parris, J., and Spacco, J. (2013) 'How we teach impacts student learning: Peer Instruction vs. lecture in cs0'. *In Proceeding of the 44th ACM Technical Symposium on Computer Science Education* 41-46

Simon, Fincher, S., Robins, A., Baker, B., Box, I., Cutts, Q., de Raadt, M., Haden, P., Hamer, J., Hamilton, M., Lister, R., Petre, M., Sutton, K., Tolhurst, D., and Tutty, J. (2006) 'Predictors of success in a first programming course'. *In Proceedings of the Eighth Australasian Computing Education Conference* (*ACE2006*) 189-196

Skudder, B. and Luxton-Reily, A. (2014) 'Worked Examples in Computer Science'. In *Proceedings of the Sixteenth Australasian Computing Education Conference (ACE2014)* 59-64

Sleeman, D. (1986) 'The challenges of teaching computer programming'. *Communications of the ACM* 29 (9), 840-841

Šmilauer, P., and Lepš, J. (2014) *Multivariate Analysis of Ecological Data using CANOCO 5*. Cambridge University Press

Smith, D. (2010) 'The Effects of Student Syndrome, Stress, and Slack on Information Systems Development Projects'. *Issues in Informing Science and Information Technology* 7, 489-494

Snell J (2011) 'Interrogating video data: systematic quantitative analysis versus micro-ethnographic analysis'. *International Journal Of Social Research Methodology* 14 (3), 253-258

Soares, A., Fonseca, F., and Martin, N. (2015) 'Teaching Introductory Programming with game design and problem-based learning'. *Issues in Information Systems* [online] 16 (3), 128-137. available from <http://www.iacis.org/iis/2015/3_iis_2015_128-137.pdf> [October 2015]

Soloway, E., and Spohrer, J. C. (1989) *Studying the Novice Programmer*. Hillsdale NJ: Lawrence Erlbaum

Sommerville, I. (2010) 9th ed. *Software Engineering*. Addison Wesley

Statista (2013) *Millennials: Best sources for entertainment in 2013, by region* [online] available from <https://www.statista.com/statistics/270958/best-source-for-entertainment-among-millenials/> [January 2019]

Stein, M. V. (2002) 'Mathematical preparation as a basis for success in CS-II'. *Journal of Computing Sciences in Colleges* 17 (4), 28-38

Stephens, M., and Rosenberg, D. (2003) *Will Pair Programming Really Improve Your Project?* [online] available from <http://www.methodsandtools.com/archive/archive.php?id=10> [September 2017]

Strategy& (2012) *This is for everyone – the case for universal digitisation* [online] available from <https://www.strategyand.pwc.com/media/file/Strategyand_This-Is-for-Everyone.pdf> [April 2017]

Sung, K., Hillyard, C., Angotti, R. L., Panitz, M. Goldstein, D. S., and Nordlinger, J. (2010) 'Game-Themed Programming Assignment Modules: A Pathway for Gradual Integration of Gaming Context Into Existing Introductory Programming Courses'. *IEEE Transactions on Education* [online] 300-304. available from <http://faculty.washington.edu/ksung/pub/2011_ieee_te-sung-2064315-proof.pdf> [June 2017]

Surbhi, S. (2016) *Difference Between Parametric and Nonparametric Test* [online] available from <http://keydifferences.com/difference-between-parametric-and-nonparametric-test.html> [November 2017]

Sweller, J. (1988) 'Cognitive Load During Problem Solving: Effects on Learning'. *Cognitive Science* 12 (2), 257-285

Sweller, J. (2010) ‘Element interactivity and intrinsic, extrinsic, and germane cognitive load’. *Educational Psychology Review* 22 (2), 123-138

Sweller, J., Ayres, P., and Kalyuga, S. (2011) *Cognitive load theory*. Springer

Sweller, J., and Cooper, G. A. (1985) ‘The Use of Worked Examples as a Substitute for Problem Solving in Learning Algebra’. *Cognition and Instruction* 2 (1), 59–89

Szulecka, T. K., Springett, N. R., and de Pauw, K. W. (1987) ‘General health, psychiatric vulnerability and withdrawal from university in first-year undergraduates’. *British Journal of Guidance and Counselling Special Issue: Counselling and Health* 15, 82-91

Taylor, J., and du Boulay, B. (1987) ‘Learning and using Prolog: an empirical investigation’. *Cognitive Science Research Reports No. 90*. Brighton: University of Sussex

Teague, D. (2011) *Pedagogy of Introductory Computer Programming: A People-First Approach*. Masters thesis. Queensland University of Technology

Teague, D. (2015) *Neo-Piagetian Theory and the novice programmer*. PhD dissertation. Queensland University of Technology

Teague, D., Corney, M., Ahadi, A., and Lister, R. (2013) ‘A Qualitative Think Aloud Study of the Early Neo-Piagetian Stages of Reasoning in Novice Programmers’. In *Proceedings of the Fifteenth Australasian Computing Education Conference (ACE2013)* 87-95. [online] available from <http://eprints.qut.edu.au/57541/1/TeagueEtAl2013aQualitativeThinkAloudStudyOfTheEarlyNeoPiagetianStagesOfReasoningInNoviceProgrammers.pdf> [June 2015]

Tech City (2015) *Tech Nation: Power the Digital Economy 2015* [online] available from <http://www.techcityuk.com/technation/> [October 2016]

Tech Nation (2018) *Report 2018: Connection and collaboration: powering UK tech and driving the economy* [online] available from <https://technation.io/wp-content/uploads/2018/05/Tech-Nation-Report-2018-WEB-180514.pdf> [January 2019]

Tech Partnership (2015) *Tech Insights: The Digital Economy, March 2015* [online] available from <https://www.thetechpartnership.com/globalassets/pdfs/research-2015/techinsights_report_mar15.pdf> [October 2016]

Teddlie, Ch., and Tashakkori, A. (2009) *Foundations of Mixed Methods Research: Integrating Quantitative and Qualitative Approaches in the Social and Behavioral Sciences.* Thousand Oaks, CA: Sage

ter Braak, C. J. F., and Prentice, I. C. (1988) ‘A theory of gradient analysis’. *Advances in Ecological Research* 18, 271-317

ter Braak, C. J. F. (1986) ‘Canonical correspondence analysis: a new eigenvector technique for multivariate direct gradient analysis’. *Ecology* 67 (5), 1167-1179

ter Braak, C. J. F. (1987) ‘Ordination’. In Jongman, R. H., ter Braak, C. J. F. and van Tongeren, O. F. R. (eds). *Data Analysis in Community Ecology*. Pudoc, Wageningen: The Netherlands

ter Braak, C. J. F. (1992) *CANOCO - a FORTRAN program for Canonical Community Ordination*. Microcomputer Power: New York

The Design Commission (2014) *Designing the digital economy: embedding growth through design, innovation, and technology* [online] available from <http://www.designcouncil.org.uk/sites/default/files/asset/document/Design%20Commission%20report%20-%20Designing%20the%20Digital%20Revolution.pdf> [December 2015]

The PISA 2003 Assessment Framework (2003) Technical report, Organisation for Economic Co-operation and Development, 2003

The Royal Society (2012) *Shut down or restart? The way forward for computing in UK schools* [online] available from <https://royalsociety.org/~/media/education/computing-in-schools/2012-01-12-computing-in-schools.pdf> [November 2015]

Thomas, K. C. (2015) *Thinking 'spatially' about mature part-time learners in HE* [online] available from <https://www.researchgate.net/publication/303523035_Thinking_'spatially'_about_mature_part-time_learners_in_HE> [March 2019]

Thomas, L. (2012a) *What Works? Student Retention and Success* [online] available from <www.heacademy.ac.uk/default/files/what_works_summary_report_0.pdf/> [April 2017]

Thomas, L. (2012b) 'Building student engagement and belonging in Higher Education at a time of change'. *Paul Hamlyn Foundation* [online] 100. available from <https://www.heacademy.ac.uk/knowledge-hub/building-student-engagement-and-belonging-higher-education-time-change-final-report> [May 2019]

Times Higher Education (2013) *Participation rates: now we are 50* [online] available from <https://www.timeshighereducation.com/features/participation-rates-now-we-are-50/2005873.article> [May 2017]

Times Higher Education (2017) *Let's have a show of hands: Times Higher Education teaching survey 2017* [online] available from <https://www.timeshighereducation.com/sites/default/files/table.pdf> [January 2020]

Tomlinson, M. (2014) *Exploring the Impacts of Policy Changes on Student Attitudes to Learning. York: Higher Education Academy* [online] available from <https://www.heacademy.ac.uk/sites/default/files/resources/Exploring_the_impact_of_policy_changes_student_experience.pdf> [May 2019]

Trafton, J. G., and Reiser, B. J. (1993) *The Contributions of Studying Examples and Solving Problems to Skill Acquisition*. PhD dissertation. Princeton University

Tukey, J. W. (1977) *Exploratory data analysis.* Addison-Wesley Series in Behavioral Science: Quantitative Methods. Reading: Addison-Wesley

Turner, R., Morrison, D., Cotton, D., Child, S., Stevens, S., Nash, P., and Kneale, P. (2017) 'Easing the transition of first year undergraduates through an immersive induction module'. *Teaching in Higher Education* [online] 22 (7), 805-821. available from <https://core.ac.uk/download/pdf/78901087.pdf> [May 2019]

Turner, C. (2018) *Universities now admitting twice as many BTEC students as they did a decade ago, figures show* [online] available from <https://www.telegraph.co.uk/education/2018/06/04/universities-now-admitting-twice-many-btec-students-dida-decade/> [July 2019]

UCAS (2010) *BTEC QCF Qualifications from 2010 (suite known as Nationals)* [online] available from <https://www.dmu.ac.uk/documents/study-documents/entry-and-admissions-criteria/generic-information/ucas-tariff-points-a-level-btec-equivalents.pdf> [Oct 2018]

UCAS (2014) *UCAS Analysis answers five key questions on the impact of the 2012 tuition fees increase in England* [online] available from <https://www.ucas.com/corporate/news-and-key-documents/news/ucas-analysis-answers-five-key-questions-impact-2012-tuition> [May 2017]

UCAS (2016a) *Applications by Detailed subject group (Applications through the main scheme)* [online] available from <https://www.ucas.com/sites/default/files/eoc_data_resource_2016-dr3_015_03.pdf> [April 2017]

UCAS (2016b) *Acceptances by subject group (All)* [online] available from <https://www.ucas.com/sites/default/files/eoc_data_resource_2016-dr3_010_01.pdf> [April 2017]

UCAS (2016c) *Progression Pathways Report (Jan 2016)* [online] available from <https://www.ucas.com/sites/default/files/progression_pathways_report_final_v2_0.pdf> [September 2017]

UCU (2016) *UCU Workload Survey 2016 Executive summary* [online] available from <https://www.ucu.org.uk/media/8196/Executive-summary---Workload-is-an-education-issue-UCU-workload-survey-report-2016/pdf/ucu_workloadsurvey_summary_jun16.pdf> [January 2020]

UK Commission for Employment and Skills (2015) *Sector insights: skills and performance challenges in the digital and creative sector* [online] available from <https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/433755/Skills_challenges_in_the_digital_and_creative_sector.pdf> [June 2016]

Universities UK (2010) *Universities and education in the post-recession economy: Report of a seminar organised by Universities UK* [online] available from <https://dera.ioe.ac.uk/26263/1/UniversitiesPostRecession.pdf> [January 2019]

Universities UK (2005) *Survey of higher education students' attitudes to debt and term-time working and their impact on attainment*. A report to Universities UK and HEFCE by the Centre for Higher Education Research and Information (CHERI) and London South Bank University. [online] available from <http://www.universitiesuk.ac.uk/policy-and-analysis/reports/Documents/2005/attitudes-to-term-time-work.pdf> [January 2019]

Universities UK (2015) *Patterns and Trends in UK Higher Education 2015* [online] available from <https://www.universitiesuk.ac.uk/policy-and-analysis/reports/Documents/2015/patterns-and-trends-2015.pdf> [January 2019]

University of Exeter (n.d) *Academic Java 1.04 - Alpha Trial* [online] available from <http://www.academicjava.com/> [September 2017]

Utting, I., Bouvier, D., Caspersen, M., Elliott-Tew, A., Frye, R., Kolikant, Y., McCracken, M., Paterson, J., Sorva, J., Thomas, L., and Wilusz, T. (2013) 'A Fresh Look at Novice Programmers' Performance and Their Teachers' Expectations'. In: *ITiCSE -WGR '13* [online] 15-32. available from <https://kar.kent.ac.uk/44804/1/ACMDLfinal-p15-utting.pdf> [May 2016]

Vaismoradi, M., Turunen, H., and Bondas, T. (2013) 'Content analysis and thematic analysis: Implications for conducting a qualitative descriptive study'. *Nursing & Health Sciences* 15 (3), 398-405

Valentine, D. W. (2004) ‘CS educational research: a meta-analysis of SIGCSE technical symposium proceedings’. *In SIGCSE ’04: Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education* 255-259

Vaughan, R. (2017) *Learning by keystrokes*. The i newspaper 9th February 2017, [online] available from <https://century2018.wpengine.com/wp-content/uploads/2018/05/Feb-2017-i-newspaper.pdf> [July 2019]

van der Veer, G., and van der Wolde, J. (1983) ‘Individual differences and aspects of control flow notations’. In Green, T. R. G., Payne, S. J., and van der Veer, D. (Eds), *The Psychology of Computer Use*. Academic Press (London)

van der Veer, G., van Beek, J., and Gruts, G. A. N. (1986) ‘Learning structured diagrams. Effects of mathematical background, instruction and problem semantics’. *Visual Aids in Programming*. Passau

van Gog, T., Kester, L., and Paas, F. (2011) ‘Effects of Worked Examples, example-problem, and problem-example pairs on novices’ learning’. *Contemporary Educational Psychology* 36 (3), 212-218

van Gog, T., Paas, F., and van Merrienboer, J. J. G. (2004) ‘Process-Oriented Worked Examples: Improving Transfer Performance Through Enhanced Understanding’. *Instructional Science* 32 (1-2), 83-98

Van Heyningen, J. J. (1997) ‘Academic achievement in college students: What factors predict success?’ *Dissertation Abstracts International Section A: Humanities & Social Sciences* 56 (6-A), 2076

Vanhanen, J., and Lassenius, C. (2007) ‘Perceived Effects of Pair Programming in an Industrial Context’. *Software Engineering and Advanced Applications* 211-218

Ventura Jr., P. R. (2005) ‘Identifying predictors of success for an objects-first CS1’. *Computer Science Education* 15 (3), 223-243

Vilner, T., and Zur, E. (2006) ‘Once She Makes It, She is There: Gender Differences in Computer Science Study’. *ITiCSE 06: Proceedings of the 11th annual conference on innovation and technology in computer science education* 227-231

Virmani, A. (2017) *How Big Data is Transforming Retail Industry* [online] available from <https://www.simplilearn.com/big-data-transforming-retail-industry-article> [June 2017]

Vygotsky, L. S. (1978) *Mind in society: The development of higher psychological processes*. Cambridge, MA: Harvard University Press

Walford, G. (1991) ‘Changing Relationship Between Government and Higher Education in Britain’, In Neave, G. and Van Vught, F. A. (eds) *The Changing Relationship Between Government and Higher Education in Western Europe,* Oxford: Pergamon Press. 165-183

Wankat, P. C., Felder, R. M., Smith, K. A., and Oreovicz, F. S. (2002) ‘The scholarship of teaching and learning in engineering’. In *Disciplinary styles in the scholarship of teaching and learning: Exploring common ground*. American Association for Higher Education and The Carnegie Foundation, Washington, DC

Watson, C., Li, F. W. B., and Godwin, J. L. (2013) ‘Predicting performance in an introductory programming course by logging and analysing student programming behaviour’. In *Proceedings of the 2013 IEEE 13th International Conference on Advanced Learning Technologies (ICALT 2013)* 319-323

Watson, C., Li, F. W. B., and Godwin, J. L. (2014) 'No tests required: comparing traditional and dynamic predictors of programming success'. *In Proceedings of the 45th ACM technical symposium on Computer science education* 469-474

Watson, C., and Li, F. W. B. (2014) 'Failure Rates in Introductory Programming Revisited'. *Innovation and Technology in Computer Science Education (ITiCSE 2014*) 39-44

Wenger, E. (1998) *Communities of Practice: Learning, Meaning and Identity*. Cambridge: Cambridge University Press

Weinberg, G. M. (1971) *The psychology of computer programming.* Van Nostrand Reinhold.

Wells, D. (2012) 'Computing in schools: time to move beyond ICT?'. *Research in Secondary Teacher Education*. [online] 2 (1), 8–13. available from <https://www.uel.ac.uk/wwwmedia/microsites/riste/David-Wells-Article-8-13.pdf> [December 2017]

Wetherell, M., Taylor, S., and Yates, S. (2001) *Discourse as data: A guide to analysis.* Sage Publications Ltd

White, G., and Sivitanides, M. (2002) 'A theory of the relationships between cognitive requirements languages and programmers' cognitive characteristics'. *Journal of Information Systems Education* 13 (1), 59-66

Whitley, Jr., B. E. (1996) 'The relationship between psychological type to computer aptitude, attitudes and behaviour'. *Computers in Human Behavior* 12 (3), 389-406

Wickham, H. (2009) *ggplot2: elegant graphics for data analysis* [online] available from <https://wiki.bioinformaticslaboratory.nl/twikidata/pub/Education/ComputinginR_new/ggplot2-book.pdf> [June 2019]

Wiedenbeck, S., Labelle, D., and Kain, V. N. (2004) 'Factors affecting course outcomes in introductory programming'. *In 16th Annual Workshop of the Psychology of Programming Interest Group* [online] 97-109. available from <http://www.ig.org/sites/default/files/2004-IG-16th-wiedenbeck.pdf> [September 2017]

Williams, L., and Kessler, R. R. (1999) 'All I Really Need to Know about Pair Programming I Learned in Kindergarten'. *Communications of the ACM.* [online] 108-114. available from <https://collaboration.csc.ncsu.edu/laurie/Papers/Kindergarten.PDF> [June 2017]

Williams, L. (2006) 'Debunking the nerd stereotype with Pair Programming'. *Computer* 39 (5), 83-85

Williams, L. (2007) 'Lessons learned from seven years of Pair Programming at North Carolina State University'. *SIGCSE Bulletin* 39 (4), 79-83

Williams, L., Kessler, R. R., Cunningham, W., and Jeffries, R. (2000) 'Strengthening the case for Pair Programming'. *IEEE Software* [online] 19-25. available from <https://collaboration.csc.ncsu.edu/laurie/Papers/ieeeSoftware.PDF> [June 2017]

Williams, L., McCrickard, D. S, Layman, L., Hussein, K. (2008) 'Eleven Guidelines for Implementing Pair Programming in the Classroom'. In *Proceedings of 2008 Agile Conference* [online] 445-452. available from <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.143.8802&rep=rep1&type=pdf> [November 2016]

Wilson, M. (1996) 'Asking questions', in Sapsford, R., and Ju, V. (eds) Data collection and analysis. Sage Publications in association with the Open University

Wilson, B. C., and Shrock, S. (2001) ‘Contributing to Success in an Introductory Computer Science Course: A Study of Twelve Factors’. *SIGCSE Bulletin* [online] 33 (1), 184-188. available from <http://delivery.acm.org/10.1145/370000/364581/p184-wilson.pdf?ip=79.69.76.228&id=364581&acc=OPEN&key=4D4702B0C3E38B35%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35%2E6D218144511F3437&__acm__=1550864420_e520c29444b709cc07be91f76330e861> [September 2017]

WISE (2014) *Women in Science, Technology, Engineering and Mathematics: The Talent Pipeline from Classroom to Boardroom UK Statistics 2014* [online] available from <https://www.wisecampaign.org.uk/uploads/wise/files/WISE_UK_Statistics_2014.pdf> [December 2016]

Wolf, A. (2011) *Review of Vocational Education – The Wolf Report* [online] available from <https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/180504/DFE-00031-2011.pdf> [December 2015]

Wood, D., and Middleton, D. (1975) ‘A study of assisted problem-solving’. *British Journal of Psychology* 66 (2), 181-191

Wood, D., Bruner, J., and Ross, G. (1976) ‘The role of tutoring in problem solving’. *Journal of Child Psychology and Child Psychiatry* 17, 89-100

Wood, E., Zivakova, L., Gentile, P., Archer, K., De Pasquale, D., and Nosko, A. (2013) ‘Examining the impact of Off-Task Multi-tasking with Technology on Real-Time Classroom Learning’. *Computers and Education* 58 (1), 365-374

Woodgates, P. (2018) Universities must innovate to adapt and succeed. [online] available from <https://www.timeshighereducation.com/hub/pa-consulting/p/universities-must-innovate-adapt-and-succeed> [July 2019]

World Economic Forum (2016) *The Future of Jobs: Employment, Skills and Workforce Strategy for the Fourth Industrial Revolution* [online] available from <http://www3.weforum.org/docs/WEF_Future_of_Jobs.pdf> [September 2016]

Woszczynski, A. B., Guthrie, T.C., Chen, T-L., and Shade, S. (2004) ‘Personality as a Predictor of Student Success in Programming Principles’. *SAIS 2004 Proceedings* [online] 1-6. available from <https://aisel.aisnet.org/sais2004/1> [March 2016]

Wu, C. C., Dale, N. B., and Bethel, L. J. (1998) ‘Conceptual models and cognitive learning styles in teaching recursion’. *SIGCSE ’98* 292-296

Yadin, A. (2011) ‘Reducing the drop out rate in an introductory programming course’. *ACM Inroads* 71-76

Yin, R. (2009) *Case Study Research Design and Methods*. 4th ed. United Kingdom: Sage Ltd

Zarb, M., Hughes, J., and Richards, J. (2012) ‘Analysing Communication Trends in Pair Programming Videos using Grounded Theory’. *Proceedings of BCS HCI 2012 Workshops Video Analysis Techniques for Human-Computer Interaction* 1-4

Zeidner, M. (1998) *Test anxiety: The state of the art*. New York, NY: Plenum

Zingaro, D., and Porter, L. (2016) ‘Impact of Student Achievement Goals on CS1 Outcomes’. *SIGCSE ’16* 279-284

Zingaro, D. (2010) ‘Experience report: Peer Instruction in remedial computer science’. In *Proceedings of Ed-Media 2010: Proceedings of the 22nd World Conference on Educational Multimedia, Hypermedia & Telecommunications (2010)* [online] 5030-5035. available from <http://www.danielzingaro.com/edmedia10.pdf> [June 2017]

Zingaro, D. (2014a) *Evaluating Peer Instruction in First-Year University Computer Science Courses*. PhD dissertation. University of Tornoto

Zingaro, D. (2014b) ‘Peer Instruction contributes to self-efficacy in CS1’*. In Proceedings of the 45th ACM technical symposium on Computer Science Education* 373-378

Zingaro, D. (2015) ‘Examining interest and performance in computer science 1: A study of pedagogy and achievement goals’. *Transactions on Computing Education* 15 (3), 1-18

Appendices

Appendix 1: Introductory Programming Course Evaluation Questionnaire

PROGRAMMING MODULE EVALUATION   Student ID ....

Please note:

(1) that this survey does not form any part of your assessed work. If you participate you may find some test results useful in evaluating your understanding of programming. Any further analysis, publication or other dissemination will be on a completely anonymous 'data' only basis. Results will also be used to inform and improve teaching practice at Bucks.

(2) that by returning the completed questionnaire you consent to your anonymous participation in this exercise.

This survey is designed to determine how effectively Ceebot and subsequent C# study has supported learning in programming. Before you start the 'test' and questionnaire please self-assess your levels of Ceebot achievement programming skills.

PLEASE TICK THE ONE MOST APPROPRIATE TO YOU
[A] I found Ceebot exercises very easy;
[B] I found Ceebot exercises moderately challenging;
[C] I found Ceebot exercises difficult but was able to understand the concepts eventually;
[D] I found Ceebot exercises very difficult and am not sure if I have understood the concepts covered.

Please complete the following test (***this is not individually assessed***) and the following questionnaire.

Question 1
Complete the following statement:
The programming construct known as a ______ describes the ordering of one programming statement or instruction after the other.

Question 2
Complete the following statement:
When it is necessary to repeat instructions programmers use a structure known as ______. Two widely used examples of this sort of structure are the ______ ______ and the ______ ______.

Question 3
Complete the following statement:
The 'if' and 'if else' statement are associated with a programming structure known as ______.

Question 4
Complete the following statement:
A ______ is a temporary storage area in a program. Before it can be used in a program it must be defined by giving it a ______ and a ______.

Question 5
For the snippet of code below …

```
int count;
for (count=0; count < 8; count ++);
{
        // statements to be repeated
}
```

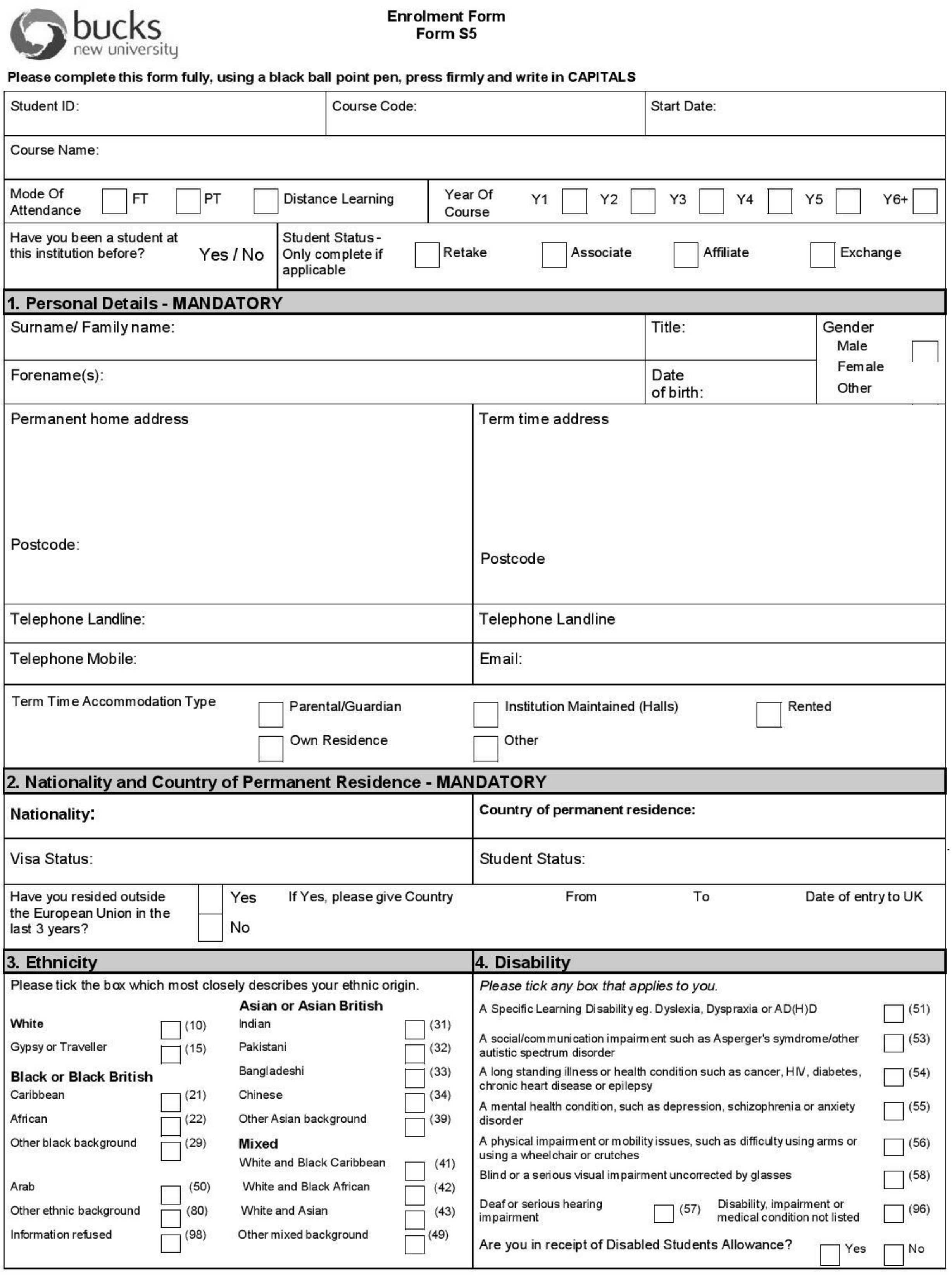
…. tick those sentences you believe to be correct:
[A] The statements will be repeated eight times.
[B] The statements will be repeated nine times, i.e. when count =0,1,2,3,4,5,6,7 and 8.
[C] The code "count ++" is the same as saying "count = count +1".
[D] The code won't work because count should be declared a "float" not an "int".
[E] The code won't work because there should not be a semicolon after "for (count=0; count < 8; count ++);".

No.	Question	Strongly agree	Agree	Neither agree nor disagree	Disagree	Strongly disagree	Comments
1	While working on exercises it is very helpful to discuss problems with friends.						
2	It is always possible to find the information required to complete Ceebot exercises.						
3	The Ceebot animated environment helps understanding of programming methods.						
4	I find it useful to draft designs and algorithms on paper before completing exercises.						
5	Ceebot does not help me remember fundamental programming concepts (e.g. structures, variables, syntax).						
6	Ceebot was enjoyable.						
7	No formal lectures are required because it is possible to understand everything from the notes.						
8	I'd like this module to be part of a commercially recognised qualification (e.g. a bit like the Cisco modules).						
9	Ceebot graphics are distracting when trying to understand programming principles.						
10	Some un-assessed multiple-choice tests included with Ceebot would help progress with learning C-type languages.						
11	It would be quicker to learn to program without Ceebot.						

No.	Question	Strongly agree	Agree	Neither agree nor disagree	Disagree	Strongly disagree	Comments
12	The easiest way to complete your logbook is to do the whole exercise (comments, algorithm, code etc.) then cut and paste everything to Word.						
13	Ceebot is a good way to begin learning the C-type programming languages required for employment.						
14	I have found that other websites sometimes have information helpful for completing exercises.						
15	I'm worried that Ceebot may not help me get a job.						
16	I only work on exercises during timetabled practical sessions.						
17	At least two hours of extra work outside practicals is required to complete the work for any one week.						
18	There are more exercises than needed to understand the concepts covered.						
19	I work on exercises at home.						
20	If it were available I'd use an electronic discussion forum to discuss problems with classmates outside practicals.						
21	Rate the two platforms we used for learning to program this year, by giving each a number in the boxes below (**1** = dreadful; **2** = poor; **3** = okay; **4** = good; **5** = great): **Ceebot** [ ] and **Visual Studio** [ ] Please write below why you preferred one platform to the other:						

<u>Please add further comments or suggestions below</u>

Appendix 2: Enrolment Form for Bucks L4 and Sri Lankan L6 Students



bucks new university

Enrolment Form
Form S5

Please complete this form fully, using a black ball point pen, press firmly and write in CAPITALS

Student ID:	Course Code:	Start Date:
Course Name:		

Mode Of Attendance	☐ FT	☐ PT	☐ Distance Learning	Year Of Course	Y1 ☐	Y2 ☐	Y3 ☐	Y4 ☐	Y5 ☐	Y6+ ☐	
Have you been a student at this institution before?	Yes / No	Student Status - Only complete if applicable	☐ Retake	☐ Associate	☐ Affiliate	☐ Exchange					

1. Personal Details - MANDATORY

Surname/ Family name:	Title:	Gender Male ☐ Female Other
Forename(s):	Date of birth:	

Permanent home address Postcode:	Term time address Postcode
Telephone Landline:	Telephone Landline
Telephone Mobile:	Email:

Term Time Accommodation Type	☐ Parental/Guardian	☐ Institution Maintained (Halls)	☐ Rented
	☐ Own Residence	☐ Other	

2. Nationality and Country of Permanent Residence - MANDATORY

Nationality:	**Country of permanent residence:**
Visa Status:	Student Status:

Have you resided outside the European Union in the last 3 years?	☐ Yes ☐ No	If Yes, please give Country	From	To	Date of entry to UK

3. Ethnicity

Please tick the box which most closely describes your ethnic origin.

White	☐ (10)	**Asian or Asian British** Indian	☐ (31)
Gypsy or Traveller	☐ (15)	Pakistani	☐ (32)
Black or Black British		Bangladeshi	☐ (33)
Caribbean	☐ (21)	Chinese	☐ (34)
African	☐ (22)	Other Asian background	☐ (39)
Other black background	☐ (29)	**Mixed**	
		White and Black Caribbean	☐ (41)
Arab	☐ (50)	White and Black African	☐ (42)
Other ethnic background	☐ (80)	White and Asian	☐ (43)
Information refused	☐ (98)	Other mixed background	☐ (49)

4. Disability

Please tick any box that applies to you.

A Specific Learning Disability eg. Dyslexia, Dyspraxia or AD(H)D	☐ (51)
A social/communication impairment such as Asperger's symdrome/other autistic spectrum disorder	☐ (53)
A long standing illness or health condition such as cancer, HIV, diabetes, chronic heart disease or epilepsy	☐ (54)
A mental health condition, such as depression, schizophrenia or anxiety disorder	☐ (55)
A physical impairment or mobility issues, such as difficulty using arms or using a wheelchair or crutches	☐ (56)
Blind or a serious visual impairment uncorrected by glasses	☐ (58)
Deaf or serious hearing impairment ☐ (57) Disability, impairment or medical condition not listed	☐ (96)
Are you in receipt of Disabled Students Allowance? ☐ Yes	☐ No

5. Parental Education - MANDATORY (For new students only)

The following question is about your parents' level of education. This includes natural parents, adoptive parents, step parents or guardians who have brought you up

Do any of your parents (as defined above) have any higher education qualifications, such as degree, diploma or certificate in higher education?

☐ Yes ☐ No ☐ Don't Know ☐ Information Refused

6. Qualifications - MANDATORY (For new students only)

New Entrants ***Please give details of HIGHEST qualifications:***
e.g. GCE A Levels / VCE A Levels / Access / HND / Nursing / FE **with grade/result**

Title	Grade	Level	Year

Most recent Institution attended *(e.g. School / FE College / Higher Education)*

Full name of institution	Month and year left
Town/City	

7. Tuition / Course Fees – MANDATORY

Tuition/Course Fees - Please tick ALL that apply

Self funding - Payment to Bucks New University	☐	(01A)
Method of Payment — In Full	☐	
In Instalments	☐	
Self funding - Payment to Partner Institution	☐	(01B)
Channel Islands/Isle of Man	☐	(06)
SLC/SAAS - Fully Funded	☐	(02)
SLC/SAAS - Partially Funded	☐	(52)
SLC/SAAS - Funding Applied For But Not Finalised	☐	(02A)
Employer Sponsor - Fully Funded (Attach Sponsor Form)	(81) ☐	
Employer Sponsor - Partially Funded (Attach Sponsor Form)	(81) ☐	
NHS	(31) ☐	
Fee Waiver - Proof Attached	☐	(05)

Tuition Fee:

Pre-registration nursing
Secondee ☐ Yes ☐ No

Post-registration nursing
NMC PIN No.

Employer :

Associate Credits

PLEASE ATTACH ALL PROOF OF FUNDING DOCUMENTS OR YOU MAY BECOME LIABLE FOR ANY TUITION/COURSE FEES

8. Student Declaration - MANDATORY

I certify that this information is correct. I agree to observe all University regulations and understand that I am ultimately responsible for payment of full prescribed fees and that formal enrolment is contingent upon this.

I agree to Buckinghamshire New University processing personal data contained in this form in accordance with the Data Protection Act 1998, and providing required information to external statutory agencies like HEFCE/HESA or its appointed agents. Further information on how your student record may be used by HESA can be found at http://bucks.ac.uk/public_information/foi/hesa_data_collection/

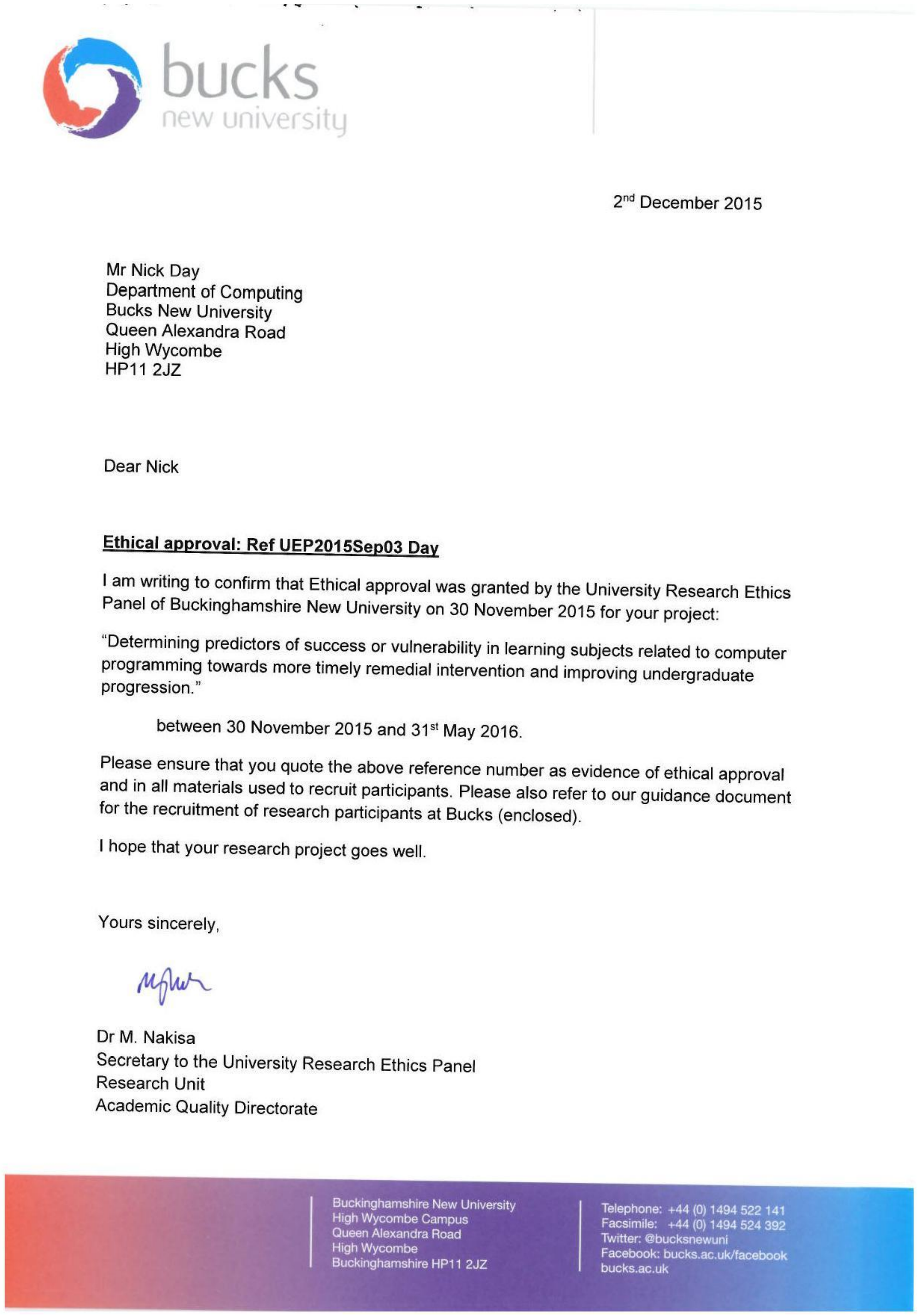
The University may use and process personal data or information about you whilst you are a student and after you have left. This may include activities to enhance the student experience (including disclosing personal data to the Bucks Students' Union and to third parties in relation to the Big Deal on Course) and automatic membership of the Bucks Alumni association after graduation.

If you have received a criminal conviction in the last 12 months please tick here: N/A ☐

Your enrolment cannot be processed if you do not sign here, or the form is incomplete. **Signature:** **Date:**

FACULTY **OFFICE USE ONLY** **Admitted by:**	**Signature** **Print name** **QUALIFICATIONS CHECKED:** **Yes** ☐ **No** ☐ Tick as appropriate	**Notes**
SC **CHECKLIST**	Passport Shown ☐ ID Card Collected ☐ Other Photographic ID Shown ☐	

Appendix 3: Ethical Approval for the collection and analysis of Bucks L4 data



bucks
new university

2nd December 2015

Mr Nick Day
Department of Computing
Bucks New University
Queen Alexandra Road
High Wycombe
HP11 2JZ

Dear Nick

Ethical approval: Ref UEP2015Sep03 Day

I am writing to confirm that Ethical approval was granted by the University Research Ethics Panel of Buckinghamshire New University on 30 November 2015 for your project:

"Determining predictors of success or vulnerability in learning subjects related to computer programming towards more timely remedial intervention and improving undergraduate progression."

between 30 November 2015 and 31st May 2016.

Please ensure that you quote the above reference number as evidence of ethical approval and in all materials used to recruit participants. Please also refer to our guidance document for the recruitment of research participants at Bucks (enclosed).

I hope that your research project goes well.

Yours sincerely,

[illegible]

Dr M. Nakisa
Secretary to the University Research Ethics Panel
Research Unit
Academic Quality Directorate

Buckinghamshire New University
High Wycombe Campus
Queen Alexandra Road
High Wycombe
Buckinghamshire HP11 2JZ

Telephone: +44 (0) 1494 522 141
Facsimile: +44 (0) 1494 524 392
Twitter: @bucksnewuni
Facebook: bucks.ac.uk/facebook
bucks.ac.uk

Appendix 4: Proposal for SAITM study

Outline for the extension to the research study:
Can student profile data explain achievement in computer programming courses?

This study proposed seeks to extend work conducted under ethics approval at Bucks. This explored the extent to which student profile data and course data available to teaching staff could explain performance in an introductory computer programming course. Analysis of first year student data from three academic years demonstrated significant differences in programming performance (grade for the module) depending on entry qualification 'route' into HE (such as A-levels and BTECs). Attendance, the number of first year modules retaken and LEA regional effects were also found to be related to programming performance.

We are keen to investigate whether these patterns of student performance are representative of other institutions that teach computer programming at undergraduate level. For this reason we seek permission to include student cohorts concerned from the partner institution, the South Asian Institute of Technology and Medicine (SAITM), and are registered on the Bucks 'top up' degree course, BSc Computing (Interactive Media).

With your permission we would like to access the data below from the Management Information Systems (MIS) database for Sri Lankan students undertaking the Advanced Interactive Programming course in 2013/14, 2014/15, 2015/16 or 2016/17:

- Attendance (Advanced Interactive Programming) (%)
- Course Name
- Local Education Authority (LEA) catchment (or equivalent: location of previous study)
- Nationality
- Ethnicity
- Gender
- Entry Qualifications
- DOB
- Advanced Interactive Programming grade
- Graphical Application Development grade
- Project grade
- Software Engineering grade
- Number of modules retaken

This data will then be analysed using the same methods as those approved for the original study at Bucks (see UEP2015Sep03). In the earlier study, profile and course data was tested for influences on student grade for computer programming modules using correlation analyses and non-parametric boxplots.

As with the earlier study (UEP2015Sep03), we will not redistribute data and will only maintain records for the duration of research. The study seeks to explore patterns that may have predictive value for guiding early pedagogic intervention and does not require individuals to be identified. Individuals will therefore remain anonymous and any publication will not contain information that may reveal a student's identity. The research team offer the assurance that all information will be treated as strictly confidential and handled in accordance with the UK Data Protection Act 1998.

Appendix 5: Ethical Approval for the collection and analysis of L6 Sri Lankan data (SAITM)

From: Azmeer Mohamed [mailto:azmeer.saitm@gmail.com]
Sent: 28 April 2017 11:58
To: Michael Everett
Subject: Re: Re-validation and continuation of BNU in Sri Lanka

Dear Mike,
This research seems interesting, I do not see any issue in participating. Please proceed.

Thanks
Az

Head, Department of New Media
Faculty of ICT & Media
South Asian Institute of Technology and Medicine
Malabe, Sri Lanka
www.SAITM.edu.lk

On Thu, Apr 27, 2017 at 4:19 PM, Michael Everett <Mike.Everett@bucks.ac.uk> wrote:
Hi Azmeer,

I have recently been contacted by one of our PhD students (Nick Day). He has asked me to enquire on his behalf, about the possibility of using certain data gained from our association with SIATM, to support his Thesis?

To this end, I would be grateful if you might care to read through his proposal, in the document attached. This outlines specifically the area he intends investigating, alongside the access to data required; relating to past and present BNU/SAITM students. (Note: all information will be used anonymously.)

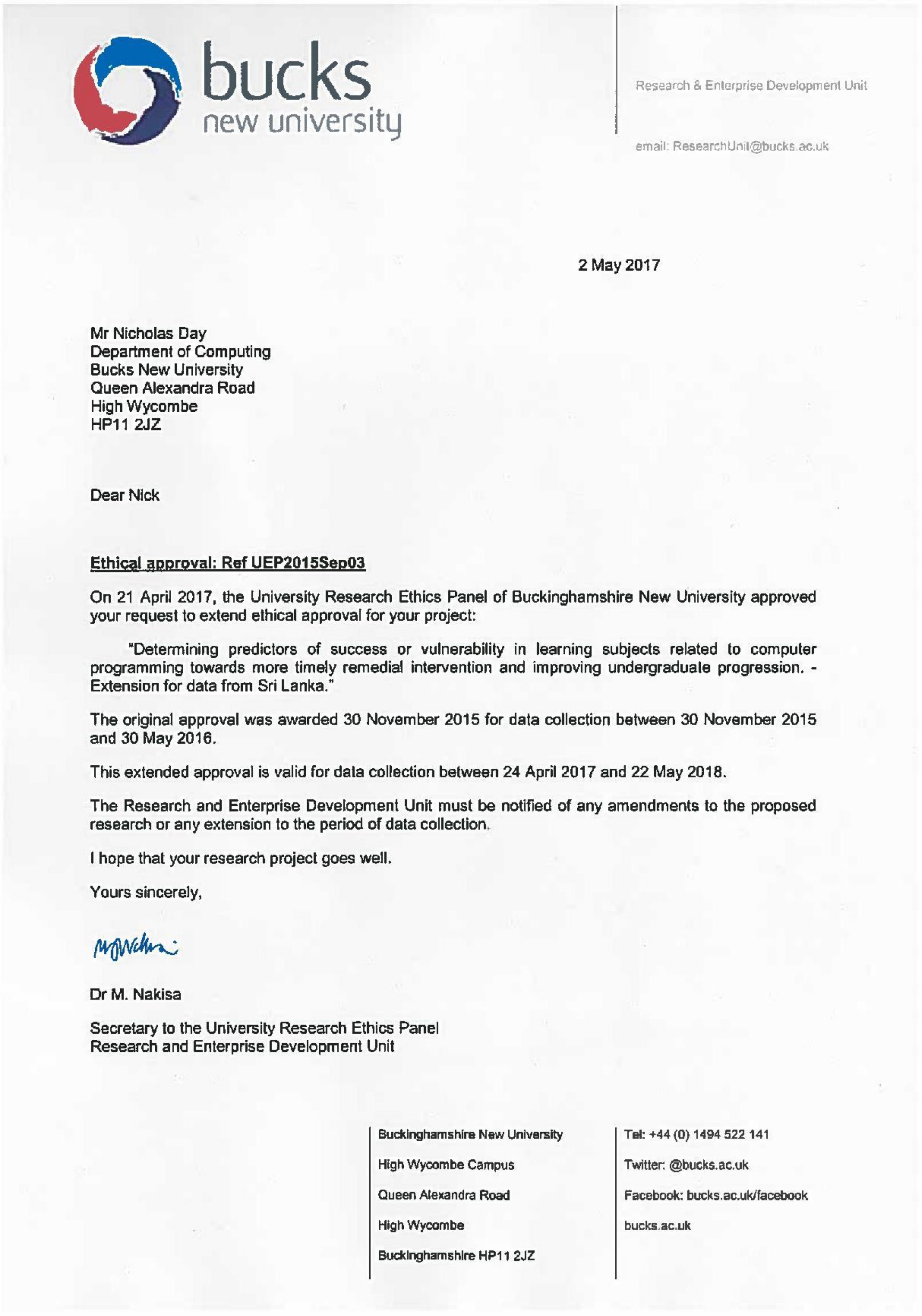
If you are in agreement with this request, then please let me know and I can give him the 'green light' to progress his research. However, clearly if you need further information or feel that you would not wish to see this pursued, then again I would ensure such wishes are addressed.

Right back to my class. Will be in contact soon.

Kind regards,

Mike

Appendix 6: Ethical Approval for the collection and analysis of L6 Sri Lankan data (Bucks)



bucks new university

Research & Enterprise Development Unit

email: ResearchUnit@bucks.ac.uk

2 May 2017

Mr Nicholas Day
Department of Computing
Bucks New University
Queen Alexandra Road
High Wycombe
HP11 2JZ

Dear Nick

Ethical approval: Ref UEP2015Sep03

On 21 April 2017, the University Research Ethics Panel of Buckinghamshire New University approved your request to extend ethical approval for your project:

"Determining predictors of success or vulnerability in learning subjects related to computer programming towards more timely remedial intervention and improving undergraduate progression. - Extension for data from Sri Lanka."

The original approval was awarded 30 November 2015 for data collection between 30 November 2015 and 30 May 2016.

This extended approval is valid for data collection between 24 April 2017 and 22 May 2018.

The Research and Enterprise Development Unit must be notified of any amendments to the proposed research or any extension to the period of data collection.

I hope that your research project goes well.

Yours sincerely,

[illegible]

Dr M. Nakisa

Secretary to the University Research Ethics Panel
Research and Enterprise Development Unit

Buckinghamshire New University
High Wycombe Campus
Queen Alexandra Road
High Wycombe
Buckinghamshire HP11 2JZ

Tel: +44 (0) 1494 522 141
Twitter: @bucks.ac.uk
Facebook: bucks.ac.uk/facebook
bucks.ac.uk

Appendix 7: Sri Lanka BTEC Interactive Multimedia unit specification

Unit Number	Unit Name	Unit Level	Unit Credits
1	Visual Communication in Art & Design	4	15
2	Idea Generation & Development in Art and Design	4	15
3	Contextual & Cultural references in Art and Design	4	15
4	Professional Practice in Art and Design	5	15
5	Project Design, Implementation and Evaluation	5	20
	Specialist units Group A (minimum 60 credits)		
63	Interactive Media Design and Prototyping	4	15
64	Interactive Media Web Authoring	5	15
65	Animation Techniques for Interactivity in Art and Design	4	15
66	Interactive Media Presentation	5	15
67	Interactive Media Principles	4	15
68	Interactive Media Teamwork	5	15
69	3D Computer Modelling and Animation	5	15
70	Computer Interface Design Principles	5	15
71	Sound Production and Editing Using Interactive Media	4	15
72	Computer Programming Principles	4	15
73	Networks and Operating Systems	4	15
74	Computer Systems Requirements Analysis	5	15
75	Website Creation and Management	5	15
76	Interactive Media Technology	4	15
77	Audio Visual Techniques in Interactive Media	4	15
78	New Technologies in Interactive Media	5	15
79	Marketing Development using Interactive Media	5	15
80	Professional Sound Production Using Interactive Media	5	15
81	Digital Video Post-Production and Editing	5	15
82	Computing Fundamentals	4	15
83	Project Management for Learning Using Interactive Media	5	15
84	2D, 3D and Time-based Digital Applications	4	15
85	Video Production	5	15
86	Digital Media in Art and Design	4	15
104	Lens-based Recording Techniques in Art and Design	4	15
111	Digital Image Creation and Development	5	15
130	Drawing Techniques and Processes in Art and Design	4	15
	Specialist units Group B (no minimum credit)		
6	Critical Study in Art and Design	5	15
7	Professional Studies in Art and Design	5	15
8	Ideas in Context	5	15
9	Research Project	5	20
12	Personal and Professional Development	5	15
13	Managing a Creative Business	4	15
14	Business Practice in Art and Design	5	15
15	Work-based Experience	5	15
16	Employability Skills	5	15
22	References and Sources in Art and Design	5	15
23	Communication with Images in Art and Design	5	15
94	Visual and Personal Presentation	4	15

Appendix 7. Level 5 BTEC Higher National Diploma (HND) in Interactive Media.
Note: The source for this data is Pearson (2015: 12 - 13).

Appendix 8: Popular Sri Lankan BTEC Interactive Multimedia units

Unit Number	Unit Name
1	Visual Communication in Art and Design
82	Computing Fundamentals
2	Idea Generation & Development in Art and Design
64	Interactive Media Web Authoring
5	Project Design, Implementation and Evaluation
4	Professional Practice in Art and Design
3	Contextual & Cultural references in Art and Design
67	Interactive Media Principles
65	Animation Techniques for Interactivity in Art and Design
69	3D Computer Modelling and Animation
76	Computer Interface Design Principles
71	Sound Production and Editing Using Interactive Media
72	Computer Programming Principles
75	Web Creation and Management
79	Marketing Development Using Interactive Media
81	Digital Video Post-Production and Editing

Appendix 8. Popular BTEC Units chosen by Sri Lankan students.

Note: This list was compiled from analysis of transcripts accompanying applications to study BSc Computing at SAITM.

Appendix 9: Information Sheet provided for Interview Participants

Information Sheet for Participants

Despite ongoing research and development of course materials, there are still challenges associated with the teaching and learning of computer programming at undergraduate level. Having taken our introductory programming module CO452, you may be aware of those challenges first hand!

A recent research study of Bucks data found patterns amongst the performance of students in CO452 and we are keen to investigate these trends in greater depth. Nick would, therefore, like to conduct a 'semi-structured' interview with you and/or your peers regarding your experiences with learning programming prior to starting your degree, as well as your working habits and interaction with peers on the course. The questions intended to be asked will be sent to you ahead of time, to give you time to consider how you might answer them. Although, other questions may arise during the conversation that are not included in this list. This interview will not be assessed in any way and has no impact on your mark for the CO452 module or any other module.

Although you may not directly benefit from the results of this research study, this is an excellent opportunity to make your voice heard and reflect on your own learning experiences. Reflecting on past learning experiences has the potential to improve your approach to the remainder of your degree course.

With your permission, Nick would like to record the conversation. This is so that he can listen again after the interview to transcribe and quote the exact words used. This recording will be treated as confidential, stored securely and will only be listened to by Nick. The recording itself and transcription of it will not be published – only select quotes which will be anonymised (unless stated otherwise). On the consent form you will also have the choice of whether you wish to make the transcribed version of your contributions available to be archived. This allows future researchers to access the transcription (not the original recording) upon request.

If you wish to participate, you have the right to withdraw at any stage during the data collection process (from now until 31st May 2017). But you will not be able to withdraw after May 2017 when the data will be analysed. The withdrawal process is easy - all you have to do is contact Nick on the email address below and let him know that you want to withdraw. You're not required to state a reason. Upon this request, your contributions will be removed from analysis and will not be published. However, we cannot guarantee that your contributions made on the recording will be destroyed. We will make our best efforts to edit out your contributions though and replace the original recording with the edited version.

Any research findings will be reported on a completely anonymous basis, unless stated on the consent form that you wish to have your name included in publications. Otherwise, names of participants will not be included in publications. Any other individuals and places mentioned during the interview will also be anonymised in transcriptions and publications.

Nick is happy to discuss any aspect of his research or this particular study with you. Please do email Nick at nicholas.day@bucks.ac.uk if you have further questions.

Appendix 10: Consent Form for Interview Participants

Consent Form for the participation in interviews for the project: 'Case studies on the culture of learning to program'

Please tick the appropriate boxes

<u>Taking part</u>

1. I have read and understood the project information sheet…………………… ☐
2. I have been given the opportunity to ask questions about the project…… ☐
3. I agree to take part in the project. Taking part in the project will include being interviewed and recorded (audio)………………………………………………… ☐
4. I understand that my taking part is voluntary; I can withdraw from the study at any time up until 31st May 2017 and I will not be asked questions about why I no longer want to take part…………………….. ☐

<u>Use of the information I provide for this project only</u>

5. *Select only one of the next two options:*

   I would like my name used where I have said or written as part of this study will be used in reports, publications and other research outputs so that anything I have contributed to this project can be recognised……………… ☐

   I do not want my name used in this project……………………………………. ☐
6. I understand my personal details such as phone number or address will not be revealed to people outside of this project……………………………………. ☐
7. I understand that my words may be quoted in publications, reports, web pages, and other research outputs but my name will not be used ***unless I requested it above***… ☐

<u>Use of the information I provide beyond this project</u>

I agree for the data I provided (transcription of interview) to be archived and made available to other researchers on request…………………………… ☐

I understand that other researchers will have access to these data only if they agree to preserve the confidentiality of these data………………………………………………… ☐

I understand that other researchers may use my words in publications, reports, web pages and other research outputs…………………………………………………………… ☐

<u>So the information you provide can be used legally</u>

I agree to assign the copyright I hold in any materials related to this project to Bucks New University……………… ☐

I consent to the processing of my personal information for the purposes of this research study. I understand that such information will be treated as strictly confidential and handled in accordance with the UK Data Protection Act 1998……………………………………………………… ☐

On this basis I am happy to participate in these interviews

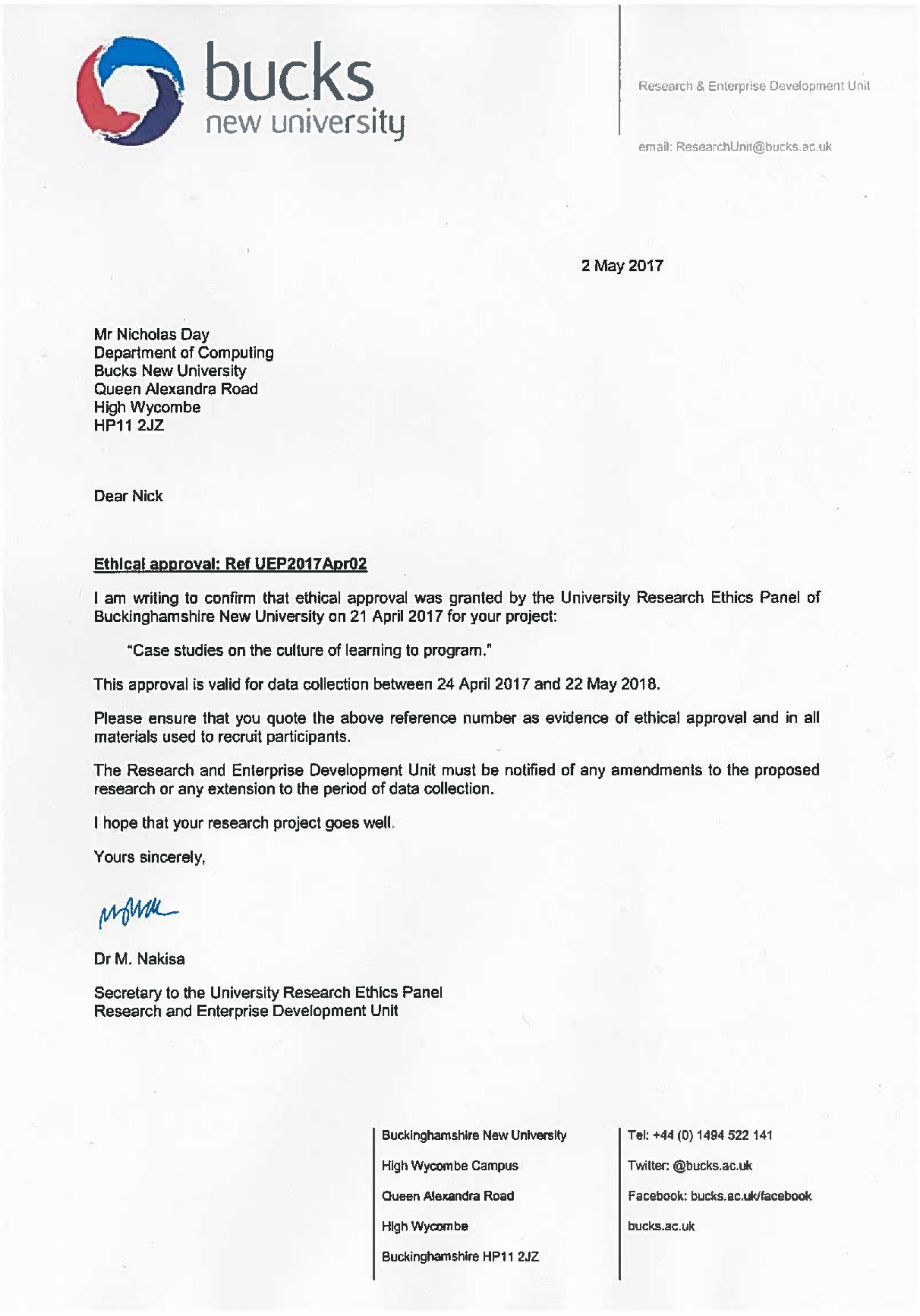
Name of Participant …………………………	Signature…………………………	Date…………
Name of Researcher………………………..	Signature…………………………	Date…………

If you have any queries or concerns, please contact: Nick Day at nicholas.day@bucks.ac.uk

One copy to be kept by the participant, one to be kept by the researcher

Adapted from the UKDA Model consent form, May16

Appendix 11: Ethical Approval for interview data collection and analysis



bucks
new university

Research & Enterprise Development Unit

email: ResearchUnit@bucks.ac.uk

2 May 2017

Mr Nicholas Day
Department of Computing
Bucks New University
Queen Alexandra Road
High Wycombe
HP11 2JZ

Dear Nick

Ethical approval: Ref UEP2017Apr02

I am writing to confirm that ethical approval was granted by the University Research Ethics Panel of Buckinghamshire New University on 21 April 2017 for your project:

"Case studies on the culture of learning to program."

This approval is valid for data collection between 24 April 2017 and 22 May 2018.

Please ensure that you quote the above reference number as evidence of ethical approval and in all materials used to recruit participants.

The Research and Enterprise Development Unit must be notified of any amendments to the proposed research or any extension to the period of data collection.

I hope that your research project goes well.

Yours sincerely,

[illegible]

Dr M. Nakisa

Secretary to the University Research Ethics Panel
Research and Enterprise Development Unit

Buckinghamshire New University
High Wycombe Campus
Queen Alexandra Road
High Wycombe
Buckinghamshire HP11 2JZ

Tel: +44 (0) 1494 522 141
Twitter: @bucks.ac.uk
Facebook: bucks.ac.uk/facebook
bucks.ac.uk

Appendix 12: UK BTEC IT Syllabus

Unit no.	Mandatory or optional unit	Unit name
1	Mandatory	Communication and Employability skills for IT
2	Mandatory	Computer Systems
3	Mandatory	Information Systems
4	Optional	Impact of the Use of IT on Business Systems
5	Optional	Managing Networks
6	Optional	Software Design and Development
7	Optional	Organisational Systems Security
8	Optional	e-Commerce
9	Optional	Computer Networks
10	Optional	Communication Technologies
11	Optional	Systems Analysis and Design
12	Optional	IT Technical Support
13	Optional	IT Systems Troubleshooting and Repair
14	**Optional**	**Event Driven Programming**
15	**Optional**	**Object Oriented Programming**
16	**Optional**	**Procedural Programming**
17	Optional	Project Planning with IT
18	Optional	Database Design
19	Optional	Computer Systems Architecture
20	Optional	Client-Side Customisations of Web Pages
21	Optional	Data Analysis and Design
22	Optional	Developing Computer Games
23	Optional	Human Computer Interaction
24	Optional	Controlling Systems Using IT
25	Optional	Maintaining Computer Systems
26	Optional	Mathematics for IT Practitioners
27	Optional	Web Server Scripting
28	Optional	Website Production
29	Optional	Installing and Upgrading Software
30	Optional	Digital Graphics
31	Optional	Computer Animation
32	Optional	Networked Systems Security
33	Optional	Supporting Business Activity
34	Optional	Business Resources
35	Specialist optional unit	Digital Graphics for Interactive Media
36	Specialist optional unit	Computer Game Platforms and Technologies
37	Specialist optional unit	2D Animation Production
38	Specialist optional unit	Interactive Media Authoring
39	Specialist optional unit	Web Animation for Interactive Media
40	Specialist optional unit	Computer Game Design
41	Specialist optional unit	3D Modelling
42	Specialist optional unit	Spreadsheet Modelling
43	Specialist optional unit	Multimedia Design

Appendix 12. Specification of units that constitute the BTEC IT Extended Diploma Award.
Notes: (1) The source for this data is (Pearson 2010a: 24); (2) Emphasis added to show units featuring programming.

Appendix 13: Previous Programming Experience Questionnaire

Previous programming experience questionnaire   Student ID ……………

Please note:

[1] This survey does not form any part of your assessed work. If you wish to participate, further analysis and dissemination of results will be on a completely anonymous 'data' only basis (no students will be named or identified). Results will be also be used to inform and improve teaching practice at Bucks.

[2] By returning this completed questionnaire you consent to your anonymous participation in this exercise.

---

1. Before starting this course did you have any previous programming experience?

   Yes / No

2. If 'yes' to Q1, please state what language(s) you have experience with below:

   ………………………………………………………………………………………

3. If 'yes' to Q1, please circle where your previous experience was (you can select more than one):

   School: a GCSE IT/Computing subject
   A-level ICT
   A-level Computing
   BTEC IT
   BTEC Games
   Previous jobs involved programming
   At home/outside of formal education
   Other (please state): ……………………………………

4. If 'yes' to Q1, in the language(s) you had experience with prior to starting the course, evaluate the
   statements below:

I knew how to output messages to the screen	Yes / No
I knew how to declare variables and assign values to them	Yes / No
I knew how to loop a set of statements	Yes / No
I knew what an 'if' statement did	Yes / No
I knew how to create functions	Yes / No
I knew how to pass values to functions	Yes / No
I knew how to return values from a function	Yes / No
I knew how to create a one dimensional array	Yes / No
I knew how to input and output the contents of an array using loops	Yes / No
I knew how to create classes and objects	Yes / No
I knew about inheritance, polymorphism, and encapsulation	Yes / No

Appendix 14: UK BTEC Games Syllabus

Unit no.	Mandatory or optional unit	Unit name
1	Mandatory	Pre-Production Techniques for the Creative Media Industries
2	Mandatory	Communication Skills for Creative Media Production
3	Mandatory	Research Techniques for the Creative Media Industries
4	Mandatory	Creative Media Production Management Project
5	Mandatory	Working to a Brief in the Creative Media Industries
13	Mandatory	Understanding the Computer Games Industry
20	**Mandatory**	**Computer Game Platforms and Technologies**
14	Optional	Working Freelance in the Creative Media Sector
15	Optional	Developing a Small Business in the Creative Media Industries
66	Optional	3D Modelling
67	Optional	3D Animation
68	Optional	3D Environments
69	Optional	Drawing Concept Art for Computer Games
70	**Optional**	**Computer Game Engines**
71	**Optional**	**Object Oriented Design for Computer Games**
72	Optional	Computer Game Design
73	Optional	Sound for Computer Games
74	Optional	Computer Game Story Development
75	Optional	Human-Computer Interfaces for Computer Games
76	Optional	Flash for Computer Games
77	Optional	Designing Tests for Computer Games
78	Optional	Digital Graphics for Computer Games

Appendix 14. Specification of units that constitute BTEC Creative Media Advertising (Games). Notes: (1) The source for this data is Pearson (2010c: 39, 49); (2) Emphasis added to show units featuring programming.

Appendix 15: UK A-level ICT Syllabus

Unit	Level	Name / topics involved	Assessment method
1	AS	Information, Systems and Applications	Exam
2	AS	Structured ICT Tasks	Exam
3	A2	ICT Systems, Applications and Implications	Exam
4	A2	ICT Project	Coursework

Appendix 15. Specification of units that constitute OCR A-level ICT.
Note: The source for this data is OCR (2013: 6).

Unit	Level	Name / topics involved	AS	AS (Double)	GCE	GCE (Double)
1	AS	The Information Age	Compulsory	Compulsory	Compulsory	Compulsory
2	AS	The Digital Economy	Compulsory	Compulsory	Compulsory	Compulsory
3	AS	The Knowledge Worker	Compulsory	Compulsory	Compulsory	Compulsory
4	AS	System Design and Installation	N/A	Compulsory	N/A	Compulsory
5	AS	Web Development	N/A	Compulsory	N/A	Compulsory
6	AS	Technical Development	N/A	Compulsory	N/A	Compulsory
7	A2	Using Database Software	N/A	N/A	Compulsory	Compulsory
8	A2	Managing ICT Projects	N/A	N/A	Compulsory	Compulsory
9	A2	Communications and Networks	N/A	N/A	N/A	Compulsory
10	A2	Using Multimedia Software	N/A	N/A	Optional	Optional
11	A2	Using Spreadsheet Software	N/A	N/A	Optional	Optional
12	A2	Customising Applications	N/A	N/A	Optional	Optional
13	A2	Web Management	N/A	N/A	N/A	Optional
14	**A2**	**Programming**	**N/A**	**N/A**	**N/A**	**Optional**

Appendix 15. Specification of units that constitute Edexcel ICT (single and double award).
Notes: (1) The source for this data is Pearson (2013: 3); (2) Emphasis added to show units featuring programming.

Appendix 16: UK A-level Computing Syllabus

Unit	GCE level	Name / topics involved	Assessment method
1	**AS**	**Problem Solving, Programming, Data Representation and Practical Exercise**	**Exam**
2	AS	Computer Components, The Stored Program Concept and The Internet	Exam
3	**A2**	**Problem Solving, Programming, Operating Systems, Databases and Networking**	**Exam**
4	**A2**	**The Computing Practical Project**	**Coursework**

Appendix 16. Specification of units that constitute AQA Computing.
Notes: (1) The source for this data is AQA (2009: 5); (2) Emphasis added to show units featuring programming.

Unit	GCE level	Name / topics involved	Assessment method
1	AS	Computer Fundamentals	Exam
2	**AS**	**Programming Techniques and Logical Methods**	**Exam**
3	**A2**	**Advanced Computing Theory (features programming)**	**Exam**
4	**A2**	**Computing Project**	**Coursework**

Appendix 16. Specification of units that constitute the OCR Computing.
Notes: (1) The source for this data is OCR (2013: 6); (2) Emphasis added to show units featuring programming.

Edexcel does not appear to have offered a Computing syllabus at A-level. It did offer a Computing GCSE, however.

Appendix 17: Information Sheet provided for participants of the video study

Information regarding the video study (CO452) to be conducted during 2016/17

It is well known that learning to program a computer can be a challenging task for many students. Much research has been conducted throughout the years in this area but little has had a significant impact on addressing the problems that students encounter. The issue appears to be complex and multifaceted. The exact nature of difficulties encountered also varies from one individual to another.

The reason why we need your help (through participating in surveys, interviews and telling us about your experiences) throughout the programming modules (CO452 and CO453) is that this research will enable us to improve our teaching and your learning experience. We hope to discover characteristics that may be early indicators of module success or where some learning guidance may be needed.

In this particular study, the video study, we want to capture how you and a friend attempt to solve a problem. By suggesting ideas to one another and thinking aloud, this will allow us to identify any common areas of the programming process that are problematic. The reason why we want to record this, rather than simply observe is that it is very easy for us (educators) to miss subtle misunderstandings and the fine detail when we are trying to help others learn how to code. By having the opportunity to watch the recording multiple times, we can identify more.

Should you wish to participate, you'll be working together on the same task, writing one solution. This models the pair programming technique, and the collaborative behaviours of the industry. We do not want you to feel any pressure to write a correct solution. That is not the aim of this study.

We aim to take recordings of pairs working through tasks throughout the 2 introductory programming modules in year 1. The future recording dates will be made known to you in advance should you wish to participate.

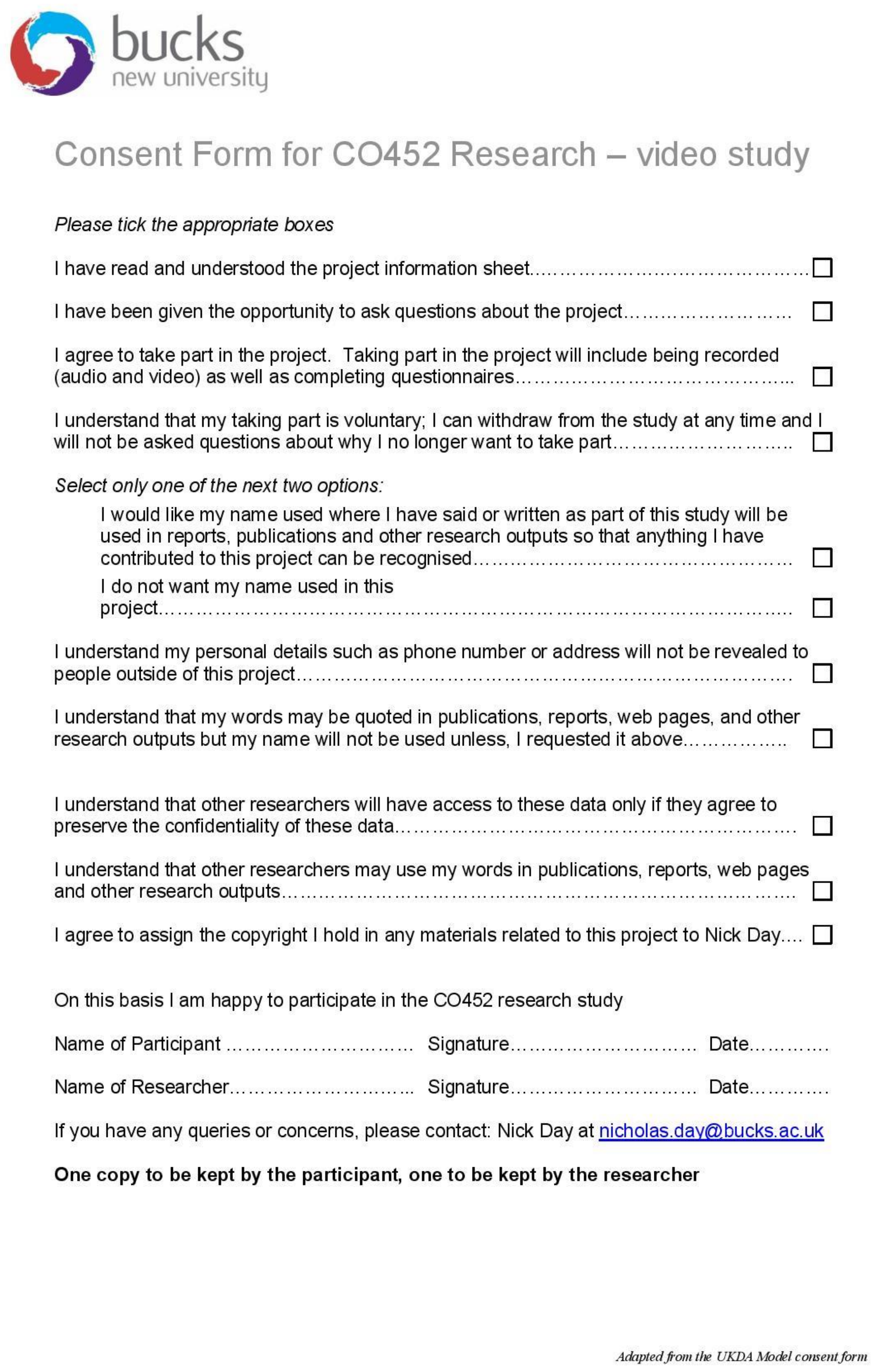
All data will be treated as confidential, stored securely and will only be viewed by Nick Day when he comes to analyse the recordings.

If you wish to participate, you have the right to withdraw at any stage during the data collection process (from now until February 2017). But you will not be able to withdraw after February 2017 when the data will be analysed. The withdrawal process is easy - all you have to do is contact Nick on the email address below and let him know that you want to withdraw. You're not required to state a reason… we honour your privacy.

Any research findings will be reported on a completely anonymous basis so that no individual may be identified. We are also very pleased to make research findings available to you should wish to know, and to discuss any aspect of our research with you.

Please do email nicholas.day@bucks.ac.uk if you have further questions.

Appendix 18: Consent Form provided for participants of the video study



bucks new university

Consent Form for CO452 Research – video study

Please tick the appropriate boxes

I have read and understood the project information sheet…………………..………………	☐
I have been given the opportunity to ask questions about the project……………………	☐
I agree to take part in the project. Taking part in the project will include being recorded (audio and video) as well as completing questionnaires………………………………...	☐
I understand that my taking part is voluntary; I can withdraw from the study at any time and I will not be asked questions about why I no longer want to take part……………………..	☐
Select only one of the next two options:	
I would like my name used where I have said or written as part of this study will be used in reports, publications and other research outputs so that anything I have contributed to this project can be recognised……………………………………………	☐
I do not want my name used in this project……………………………………………………………………….	☐
I understand my personal details such as phone number or address will not be revealed to people outside of this project………………………………………………………………….	☐
I understand that my words may be quoted in publications, reports, web pages, and other research outputs but my name will not be used unless, I requested it above……………..	☐
I understand that other researchers will have access to these data only if they agree to preserve the confidentiality of these data………………………..…………………………….	☐
I understand that other researchers may use my words in publications, reports, web pages and other research outputs……………………………..…………………………..…….	☐
I agree to assign the copyright I hold in any materials related to this project to Nick Day….	☐

On this basis I am happy to participate in the CO452 research study

Name of Participant ………………………… Signature………………………… Date………….

Name of Researcher……………………….. Signature………………………… Date………….

If you have any queries or concerns, please contact: Nick Day at nicholas.day@bucks.ac.uk

One copy to be kept by the participant, one to be kept by the researcher

Adapted from the UKDA Model consent form

Appendix 19: Task description for S1 and S2

Project Task: Traffic Survey 3 (do not press F1 as the instructions are different)
There is a very busy road near the control centre and the weather is dangerously foggy at this time of year. Your task is to record the traffic flow using a programmed robot stationed across the road from the control centre.
There are 6 different kinds of robots on the road. How can we detect one of many kinds?
The answer is to use an array.

Survey Requirements
The traffic is, of course, travelling in 2 directions. You are required to count the clockwise traffic and the anti-clockwise traffic.

You should display:

- The category of each robot as it is detected
- The total count of clockwise and anticlockwise traffic

Program Guidance
You should use your robot's radar to help you. The traffic consists of **TrackedGrabber** robots, **TrackedShooter** robots, **WheeledGrabber** robots, **LeggedGrabber** robots, L**eggedShooter** robots and **WheeledShooter** robots. You need to use the radar to detect all of these. However, you will need to use a narrow radar detection beam otherwise you will detect robots that are not directly in front of your robot.
Using the instruction: item=radar (…… , 0,4); will make your robot look ahead using a 4 degree beam. But what do you use if you have 6 possible robots to detect? Read on! You will need to use a loop and also some way of timing the counting process. The function abstime() is useful as it tells you how long the task has been running.

Note: clockwise traffic can be detected from 8 to 20 metres away from the robot, and anti-clockwise traffic can be detected from 0 to 8 metres from the robot.
You can use 2 radar beams set using these distances, e.g.:

```
nearRobot = radar(list, 0, 4, 0, 8);
farRobot = radar(list, 0, 4, 8, 20);
```

Appendix 20: Task description for S3 and S4

1.6 Adding a Splash Screen to the Project

- Suppose we want to add a new form to our project to display a startup screen when we first run our program.
- This is called a **Splash Screen**
- When we click this form it should load and display the second calculator form that we have just designed. Here goes!

Adding a New Form

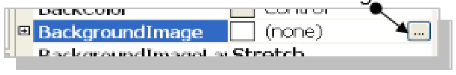
- Select **Project** from the menubar, followed by **Add Windows Form**
- Select the **Windows Form** icon and type a name for the form: **frmSplash**
- Then click the **Add** button

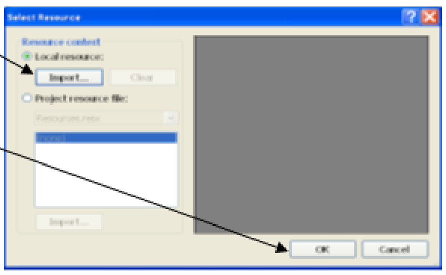
Designing the Form

- Change the **FormBorderStyle** property to **None**
- Add a label to the form and set its Text property to **Click to use Calculator**
- Increase the **Font Size** to 14
- Set the **StartPosition** property to **CenterScreen**

Adding a Background Image

- Select the **BackgroundImage** property and click the little button on the right
- In the **Select Resource** window, Click the **Import** button and find a suitable picture (try CalcHand.jpg, CalcMan.jpg or CalcGirl.jpg) and **Open** it.
- Then click the **OK** button
- Finally find the **BackgroundImageLayout** property and change it to **Stretch**.
- Then adjust the size of your form so your background looks good.





Adding Code to Calculator Form

We want the **frmSplash** form to be displayed before the Calculator form, **frmCalc**. There are several ways to do this. We shall use the **Load** event which takes place as a form loads .. we shall use it to show the **frmSplash** form:

- Open the calculator form (Form1) in Design mode
- Double-click on the form background (not one of the controls)
    - You should find you are in the **Form1_Load()** method
- Add the following code to this event :

```
frmSplash  SScreen = new frmSplash();   // create new SScreen form object
SScreen.Show();                         // show the splash screen
System.Threading.Thread.Sleep(2000);    // pause for 2 seconds
SScreen.Close();                        // close the splash screen
```

- Now run the project and the splash screen should show for 2 seconds before closing, when the Calculator should load up in the normal way.

Click Splash Screen to start

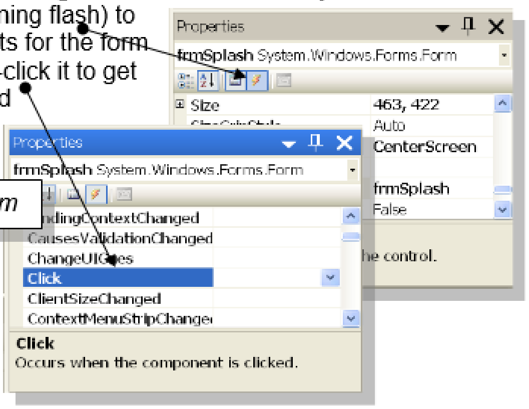
- The splash screen at the moment shows for 2 seconds. How can we get it to show until we decide to click it? We need to change a few things.
- Change the code in the **Form1_Load()** method:

```
frmSplash  SScreen = new frmSplash();  // create new SScreen form object
SScreen.ShowDialog();                  // show the splash screen
```

- The **ShowDialog()** method is used here instead of **Show()** .. it shows the form until you decide to close it. But how can we close the form?
- Now look at the **frmSplash** form **Design** and examine the **Properties** window
- Click on the **Events** button (lightning flash) to see a list of all the possible events for the form
- Find the **Click** event and double-click it to get to the **frmSplash_Click()** method
- Add code to close down the current form:

```
this.Close();   // close current form
```

- Now run the project again and your splash screen should work perfectly!



Task 1.6: Splash Screens

1. Complete the splash screen so that it works with a 2 second delay
2. Then complete the **Click to use Calculator** style of splash screen

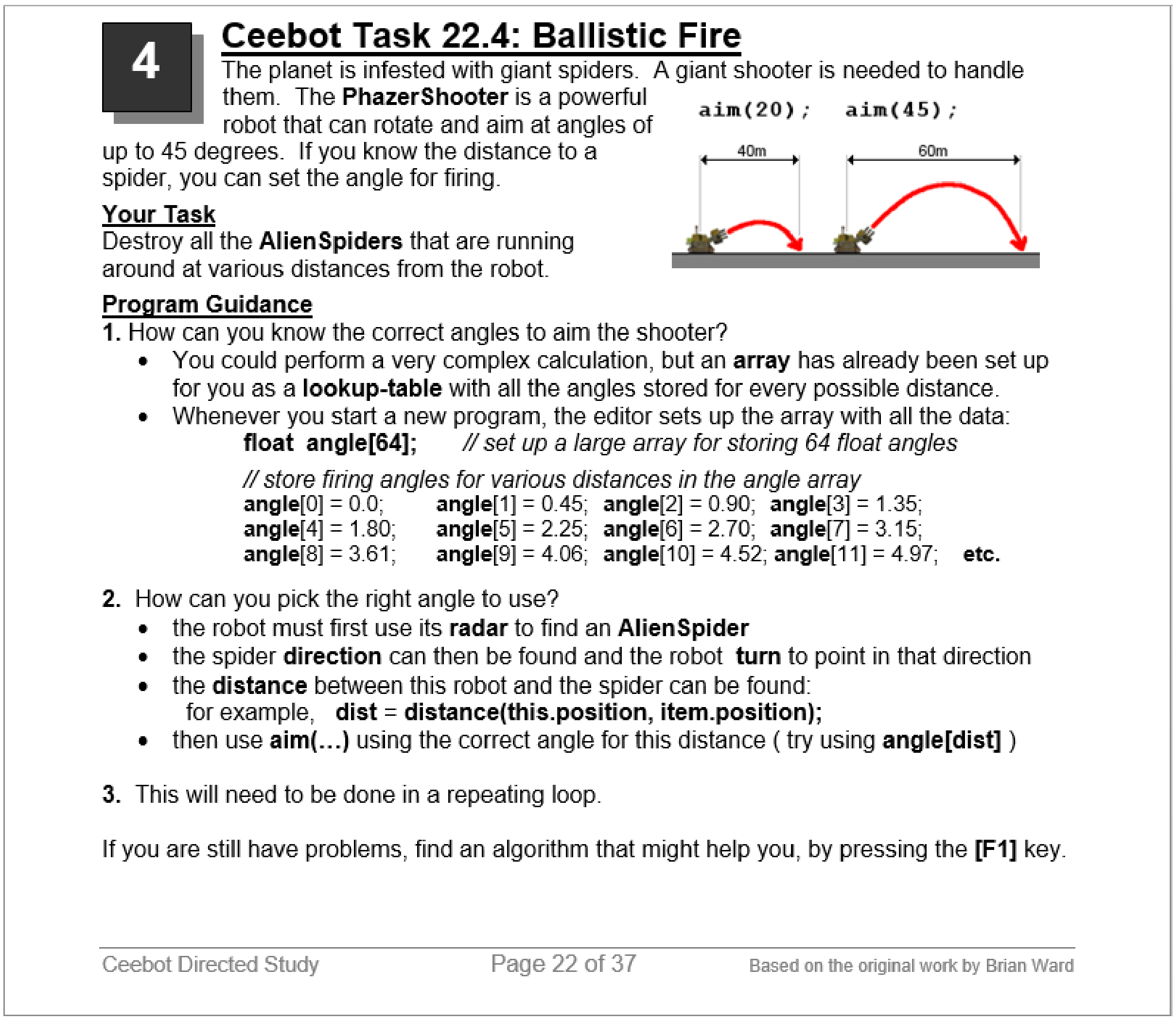
Task 1.7: Quit Screen

1. Create a new form called **frmQuit** which has a nice friendly GoodBye message, and a **Click to Finish** message and a suitable background picture.
2. When you click the calculator's **Quit** button it should display your new form **CenterScreen** and then finish when you click it.

Task 1.8: More Calculator Buttons

1. Add another button to your calculator, **Power** that takes the first number and raises it to the power of the second number using the **Math.Pow(-- , --)** function.
   - To test it : **3** to the power of **4** is **81** (3 x 3 x 3 x 3)
     4 to the power of **3** is **64** (4 x 4 x 4)
2. Add another button called **Average** that works out the average of the 2 numbers entered in the text boxes
3. Create another button called **Clear** that clears all values entered in the text boxes.

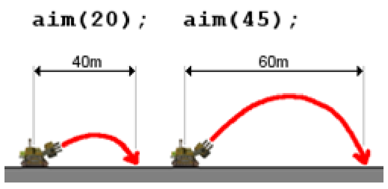
Appendix 21: Task description for S5 and S6, and S7 and S8



4

Ceebot Task 22.4: Ballistic Fire

The planet is infested with giant spiders. A giant shooter is needed to handle them. The **PhazerShooter** is a powerful robot that can rotate and aim at angles of up to 45 degrees. If you know the distance to a spider, you can set the angle for firing.



Your Task

Destroy all the **AlienSpiders** that are running around at various distances from the robot.

Program Guidance

1. How can you know the correct angles to aim the shooter?

- You could perform a very complex calculation, but an **array** has already been set up for you as a **lookup-table** with all the angles stored for every possible distance.
- Whenever you start a new program, the editor sets up the array with all the data:

```
float  angle[64];      // set up a large array for storing 64 float angles

// store firing angles for various distances in the angle array
angle[0] = 0.0;     angle[1] = 0.45;  angle[2] = 0.90;  angle[3] = 1.35;
angle[4] = 1.80;    angle[5] = 2.25;  angle[6] = 2.70;  angle[7] = 3.15;
angle[8] = 3.61;    angle[9] = 4.06;  angle[10] = 4.52; angle[11] = 4.97;   etc.
```

2. How can you pick the right angle to use?

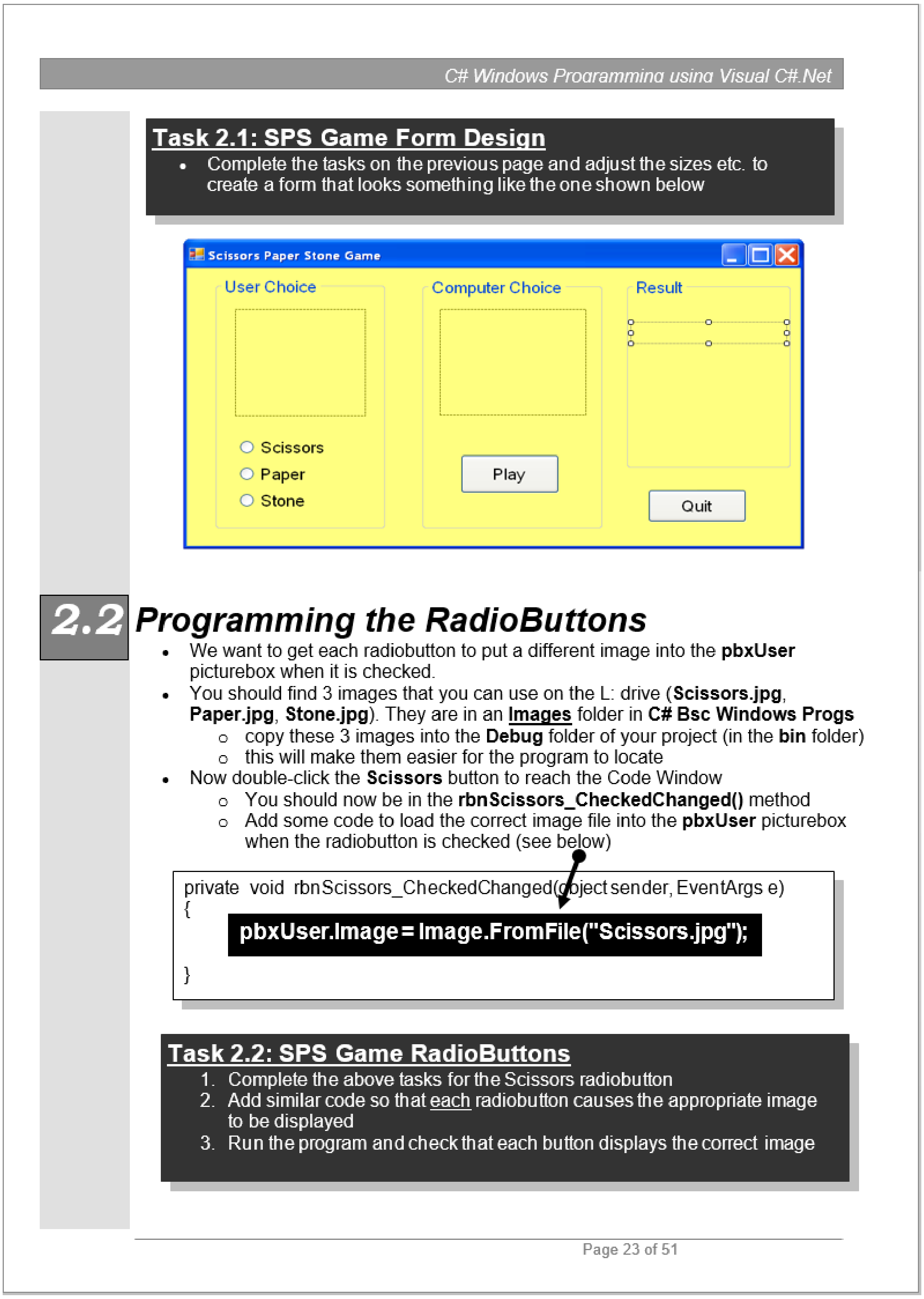
- the robot must first use its **radar** to find an **AlienSpider**
- the spider **direction** can then be found and the robot **turn** to point in that direction
- the **distance** between this robot and the spider can be found:
  for example, **dist** = **distance(this.position, item.position);**
- then use **aim(…)** using the correct angle for this distance (try using **angle[dist]**)

3. This will need to be done in a repeating loop.

If you are still have problems, find an algorithm that might help you, by pressing the **[F1]** key.

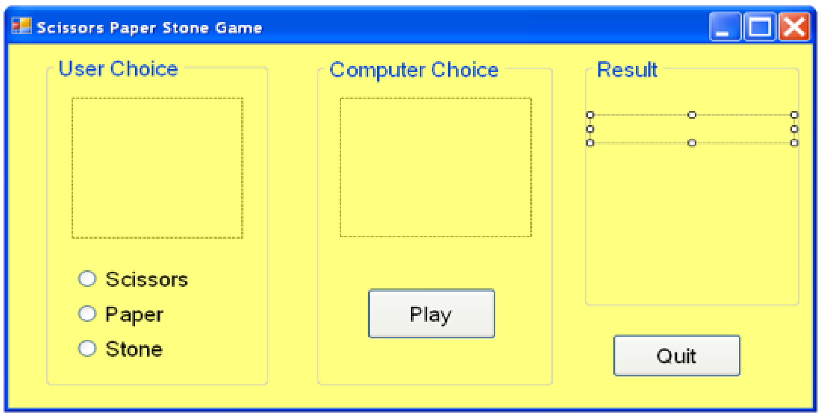
Ceebot Directed Study | Page 22 of 37 | Based on the original work by Brian Ward

Appendix 22: Task description for S9 and S10



Task 2.1: SPS Game Form Design

- Complete the tasks on the previous page and adjust the sizes etc. to create a form that looks something like the one shown below



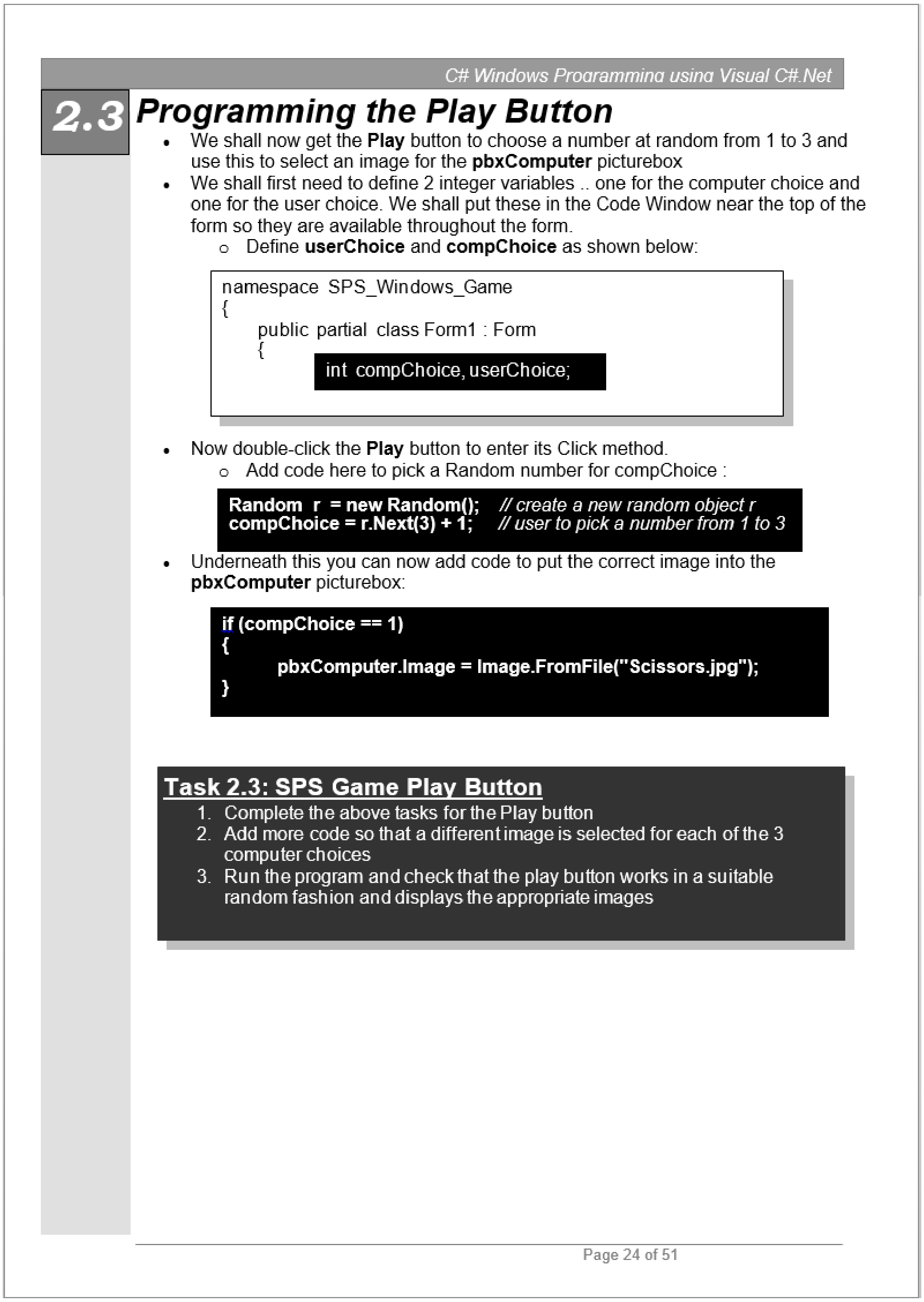
2.2 *Programming the RadioButtons*

- We want to get each radiobutton to put a different image into the **pbxUser** picturebox when it is checked.
- You should find 3 images that you can use on the L: drive (**Scissors.jpg**, **Paper.jpg**, **Stone.jpg**). They are in an **Images** folder in **C# Bsc Windows Progs**
  - copy these 3 images into the **Debug** folder of your project (in the **bin** folder)
  - this will make them easier for the program to locate
- Now double-click the **Scissors** button to reach the Code Window
  - You should now be in the **rbnScissors_CheckedChanged()** method
  - Add some code to load the correct image file into the **pbxUser** picturebox when the radiobutton is checked (see below)

```
private void rbnScissors_CheckedChanged(object sender, EventArgs e)
{
        pbxUser.Image = Image.FromFile("Scissors.jpg");
}
```

Task 2.2: SPS Game RadioButtons

1. Complete the above tasks for the Scissors radiobutton
2. Add similar code so that each radiobutton causes the appropriate image to be displayed
3. Run the program and check that each button displays the correct image



2.3 Programming the Play Button

- We shall now get the **Play** button to choose a number at random from 1 to 3 and use this to select an image for the **pbxComputer** picturebox
- We shall first need to define 2 integer variables .. one for the computer choice and one for the user choice. We shall put these in the Code Window near the top of the form so they are available throughout the form.
    - Define **userChoice** and **compChoice** as shown below:

```
namespace  SPS_Windows_Game
{
      public  partial  class Form1 : Form
      {
                int  compChoice, userChoice;
```

- Now double-click the **Play** button to enter its Click method.
    - Add code here to pick a Random number for compChoice :

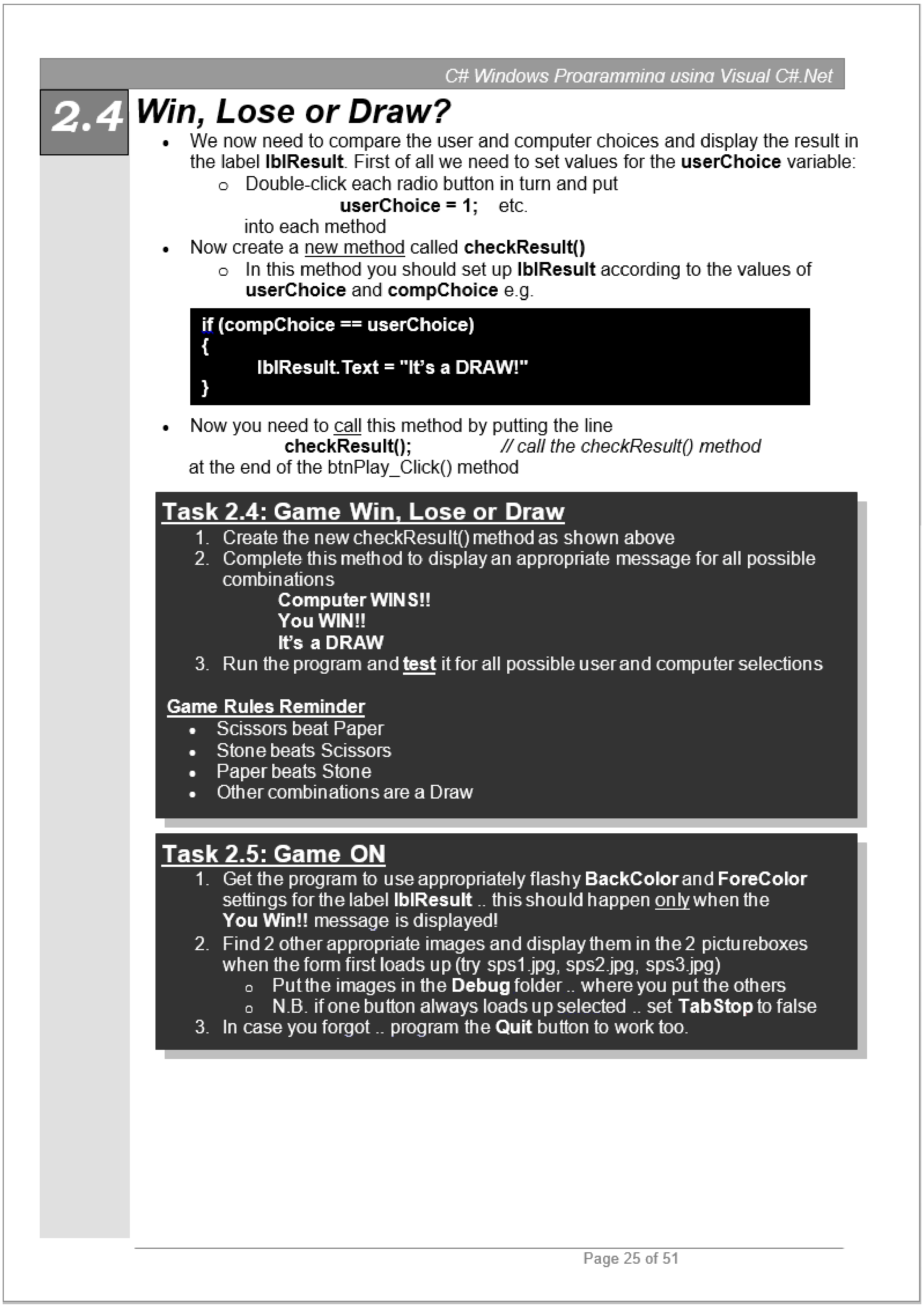
```
Random  r  = new Random();    // create a new random object r
compChoice = r.Next(3) + 1;    // user to pick a number from 1 to 3
```

- Underneath this you can now add code to put the correct image into the **pbxComputer** picturebox:

```
if (compChoice == 1)
{
        pbxComputer.Image = Image.FromFile("Scissors.jpg");
}
```

Task 2.3: SPS Game Play Button

1. Complete the above tasks for the Play button
2. Add more code so that a different image is selected for each of the 3 computer choices
3. Run the program and check that the play button works in a suitable random fashion and displays the appropriate images



2.4 *Win, Lose or Draw?*

- We now need to compare the user and computer choices and display the result in the label **lblResult**. First of all we need to set values for the **userChoice** variable:
  - Double-click each radio button in turn and put

    userChoice = 1; etc.

    into each method
- Now create a new method called **checkResult()**
  - In this method you should set up **lblResult** according to the values of **userChoice** and **compChoice** e.g.

```
if (compChoice == userChoice)
{
        lblResult.Text = "It's a DRAW!"
}
```

- Now you need to call this method by putting the line

  checkResult(); *// call the checkResult() method*

  at the end of the btnPlay_Click() method

Task 2.4: Game Win, Lose or Draw

1. Create the new checkResult() method as shown above
2. Complete this method to display an appropriate message for all possible combinations

   Computer WINS!!
   You WIN!!
   It's a DRAW
3. Run the program and **test** it for all possible user and computer selections

Game Rules Reminder

- Scissors beat Paper
- Stone beats Scissors
- Paper beats Stone
- Other combinations are a Draw

Task 2.5: Game ON

1. Get the program to use appropriately flashy **BackColor** and **ForeColor** settings for the label **lblResult** .. this should happen only when the **You Win!!** message is displayed!
2. Find 2 other appropriate images and display them in the 2 pictureboxes when the form first loads up (try sps1.jpg, sps2.jpg, sps3.jpg)
   - Put the images in the **Debug** folder .. where you put the others
   - N.B. if one button always loads up selected .. set **TabStop** to false
3. In case you forgot .. program the **Quit** button to work too.

Appendix 23: Week 4 Formative Quiz questions for 2015/16

1. Multiple Choice: What are the three main programming constructs?

- Sequence, Selection, Iteration
- Sequence, Selection, Increment
- Sequence, Selection, Implementation
- Sequence, Selection, Invoke

2. True / False: Using the AND operator to test two or more conditions means that either one of the conditions could be true for the test condition to be true

- True
- False

3. Multiple Choice: What would be the output of the code below:

```
int a = 20;
int b = 10;
a = b;
message(a);  //output variable a
```

- 10
- 20
- 0
- 2010

4. Multiple Choice: What does the == operator do?

- Assigns a value to a variable
- Compares two values
- Increases the count by 1
- Assigns the value twice

5. Multiple Choice: A type that has only has two possible values (true and false) is known as...

- Boolean
- String
- Integer
- Float

6. True / False: Will the following code result in an infinite loop?

```
int count = 0;
int number = 3;
while (count < 10){
        total = total * number;
}
count++;
message("the total is : " + total);
```

- True
- False

7. Multiple Choice: The 'if' statement is an example of what programming construct

- Selection
- Sequence
- Iteration
- Subsystem

8. Multiple Answer: Which of the following are examples of iterative statements

- for
- do... while
- while
- switch

9. Multiple Choice: Variables need two things to be declared (in most strongly typed languages):

- A ‘type’ and a ‘name’
- A ‘function’ and a ‘name’
- A ‘type’ and a ‘function’
- A ‘selection’ and a ‘type’

10. True / False: Using the suffix '++' (e.g. 'count++') would increment the value of the variable by 1

- True
- False

11. Multiple Choice: What does concatenation mean?

- The sum of a set of numbers
- Assigning a value to a variable
- Another word for joining data together
- Another word for iterating through statements

12. True / False: When we assign values to variables, these values are stored at a memory address on the hard drive.

- True
- False

13. Multiple Choice: How would you calculate the average of 6 numbers added together?

- Divide the sum of the 6 numbers by 6
- Multiple the sum of the 6 numbers by 6
- Add 6 to the sum of the 6 numbers
- Subtract 6 from the sum of the 6 numbers

14. Multiple Choice: What will be the output of the code below

```
int firstNum = 10;
int secNum = 20;

message(“This program will test to see if two numbers are in range”);

if (firstNum <=10 && secNum <= 10){
        message (“These numbers are in range”);
}
else{
        message(“These numbers are not in range”);
}
```

- This program will test to see if two numbers are in range
  These numbers are in range

- This program will test to see if two numbers are in range
  These numbers are not in range

- This program will test to see if two numbers are in range

- These numbers are not in range

15. Multiple Choice: Declaring a variable and assigning a value in the same line is an example of…

- Initialisation
- Incrementing
- Iteration
- Invocation

16. Multiple Choice: Writing a ‘for loop’ within a ‘for loop’ is an example of...

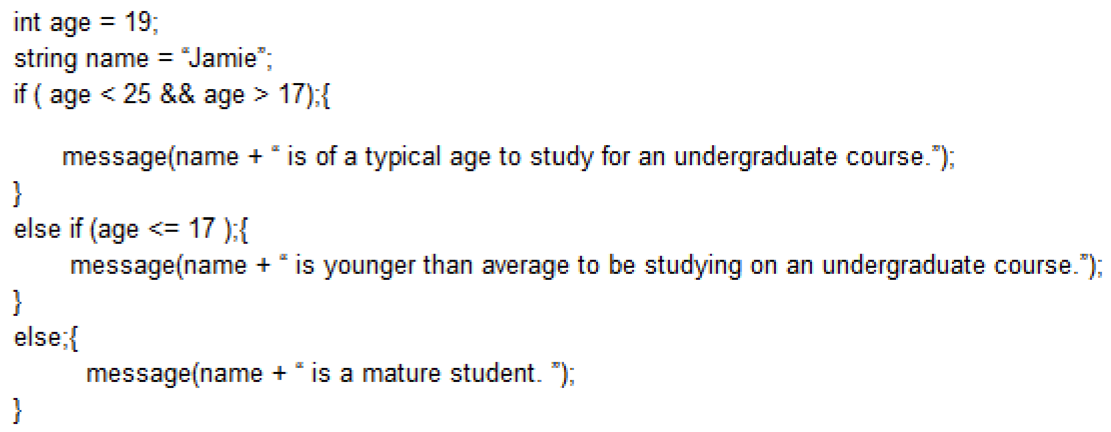
- Nesting
- Looping infinitely
- Repeating the statements twice
- Conditional looping

17. True / False: True or false: the code in the ‘if’ statement below will be executed.

```
string name;
name = “nicholas”;
if(name = “nicholas”){
        message(“Pleased to meet you, “ + name);
}
```

- True
- False

18. Multiple Choice: What will the output be below:



```
int age = 19;
string name = "Jamie";
if ( age < 25 && age > 17);{

      message(name + " is of a typical age to study for an undergraduate course.");
}
else if (age <= 17 );{
       message(name + " is younger than average to be studying on an undergraduate course.");
}
else;{
         message(name + " is a mature student. ");
}
```

- Jamie is of a typical age to study for an undergraduate course.
- Jamie is younger than average to be studying on an undergraduate course.
- Jamie is a mature student.
- Nothing will be output

19. Multiple Choice: How many statements are contained within the ‘for' loop parentheses?

- 1
- 2
- 3
- 4

20. Multiple Choice: What symbols would you write to use the OR operator?

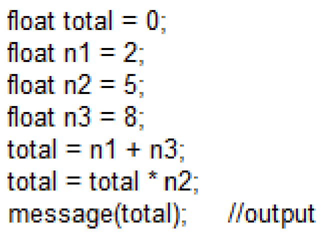
- ||
- &&
- %%
- ££

Appendix 24: Week 9 Formative Quiz questions for 2015/16

1. Multiple Choice: What are the three main programming constructs?

- Sequence, Selection, Iteration
- Sequence, Selection, Incrementing
- Sequence, Selection, Implementation
- Sequence, Selection, Invoke

2. Multiple Choice: What would be the output of the code below:



```
float total = 0;
float n1 = 2;
float n2 = 5;
float n3 = 8;
total = n1 + n3;
total = total * n2;
message(total);     //output
```

- 15
- 50
- 56
- 0

3. Multiple Choice: What does the == operator do?

- Assigns a value to a variable
- Compares two values to see if they are equal
- Increases the count by 1
- Assigns the value twice

4. Multiple Choice: Variables need two things to be declared (in most strongly typed languages):

- A 'type' and a 'name'
- A 'function' and a 'name'
- A 'type' and a 'function'
- A 'selection' and a 'type'

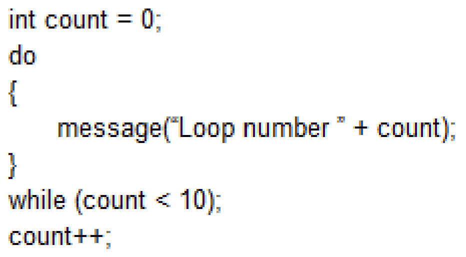
5. Multiple Answer: Which of the following are examples of iterative statements

- for
- do... while
- if
- switch

6. Multiple Choice: What does concatenation mean?

- Another word for joining data together
- Another word for iterating through statements
- Assigning a value to a variable
- The sum of a set of numbers

7. Multiple Choice: How many times will the loop below run?



```
int count = 0;
do
{
      message("Loop number " + count);
}
while (count < 10);
count++;
```

- 0
- 1
- 10
- infinite

8. True / False: Ceebot (and other strongly typed languages) will allow you to store string data in integer variables.

- True
- False

9. Multiple Choice: Declaring a variable and assigning a value in the same line is an example of...

- Initialisation
- Incrementing
- Iteration
- Invocation

10. True / False: The OR operator requires both conditions (or all conditions, if more than 2) that are being checked to be true, in order to make the whole condition true.

- True
- False

11. Multiple Choice: When working with arrays, what is the technical name for the number that is used to refer to a particular element:

- Index
- Reference
- Value
- Position

12. Multiple Choice: What is the value of the integer variable ‘c’ in the code below:

```
extern void object::test()
{
    int a = 9;
    int b = 10;
    multiply(a,b);
}
void object::multiply(int c, int d)
{
    int result = c * d;
    message(“The result of multiplying “ + c + “ with “ + d + “ is “ + result); //output to screen

}
```

- 90
- 9
- 0
- 10

13. True / False: Passing parameters by-value means that the value of the actual parameter is assigned to the formal parameter

- True
- False

14. Multiple Choice: What is the value held in the 3rd element of the array below:

```
int A[5];
for (int i = 0; i < 5; i++){
            A[i] = i * 5;
}
```

- 0
- 5
- 10
- 15

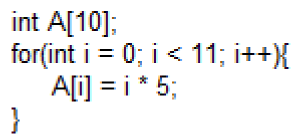
15. True / False: User defined functions don't necessarily have to return the same type of data that it is being passed to it (if it is being sent data at all).

- True
- False

16. True / False: In Ceebot (and other strongly typed languages) arrays can only store data of the same type.

- True
- False

17. True / False: The code below is syntactically and logically correct:



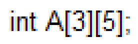
```
int A[10];
for(int i = 0; i < 11; i++){
     A[i] = i * 5;
}
```

- True
- False

18. Multiple Choice: What does the number contained within the square brackets represent when an array is declared?

- The number of elements that the array will have
- The number of arrays that are being initialised
- It indicates that this will be a dynamic array
- The number of dimensions that the array will have

19. Multiple Choice: How many dimensions are being specified in the declaration below:



```
int A[3][5];
```

- 1
- 2
- 8
- 15

20. Multiple Choice: To pass an array from one function to another you have to pass the name of the array as a parameter. What does the name of the array hold?

- The memory address of the first element
- The value of the first element
- All the values of all the elements stored in the array
- The value of the last element of the array