



SCHOOL OF DESIGN, ENGINEERING  
& COMPUTING

MSc Enterprise Information Systems  
September 2013

Computing in Education: A study of computing in education  
and ways to enhance students' perceptions and understanding  
of computing

by

Paul Albinson BSc (Hons), FdSc, MBCS

## Abstract

There is a huge demand for computing skills in industry due to computing becoming ubiquitous and essential for modern life. Yet despite this, industry struggles to find employees with suitable computing skills and similarly Further and Higher Education institutions have observed a lack of interest in their computing courses in recent years.

This study looks at possible reasons for this lack of interest in computing, how computing is taught in education and ways to improve students' perceptions and understanding of computing. It focuses around a case study of a university outreach event for secondary schools which investigated how interactive teaching methods can be used to enhance students' perceptions and understanding of computing and to increase their computing knowledge. It includes the use of physical computing and was designed to make computing fun, motivational and relevant, and to provide examples of real-world applications. Surveys were used before and after the event to understand what students' impressions and knowledge of computing is and to see if the event improved these. Observations were also used to see how well the students handled the event's content and whether they appeared to enjoy and understand it.

Results from the case study indicate that interactive teaching methods enhance computing education, and physical computing with electronics can enhance lessons and show the relevance of computing with examples of real-world applications, and can be fun and motivational. The case study provides teachers with example tasks and challenges they can use with their students and/or ideas around other interactive teaching methods including practical computing.

## Acknowledgements

The author would like to thank Dr Sheridan Jeary for the guidance, supervision and encouragement she provided during this project. The author is also grateful for St Edward's School hosting the pilot event/case study and especially to Alastair Barker for his advice and dedication to the event as well as advice on the dissertation and insights into school computing teaching. The author also wishes to thank the students and staff involved in the event for their valuable contribution. Thanks also goes to Kane Lean for his advice on creating an outreach school event and details on school computing teaching. The author also thanks Stuart Wray for participating in the initial discussion which provided many ideas for the dissertation including the idea for the pilot event. Thanks also go to Jules Thompson for providing details on school computing teaching.

## Contents

Abstract .....	1
Acknowledgements .....	3
Figures.....	9
Tables .....	11
Glossary .....	13
1. Introduction.....	15
2. Background to the project .....	16
2.1. English university enrolments.....	16
2.2. Computing in schools.....	19
2.3. The new National Curriculum.....	20
2.4. Implementing the new National Curriculum.....	21
2.5. Teaching programming .....	22
2.6. Teaching computer concepts.....	23
3. Comparing programming languages/tools/environments .....	24
3.1. Visual programming languages/tools/environments.....	25
3.2. Text-based programming languages .....	25
3.2.1. Python .....	25
3.2.2. Logo .....	26
3.2.3. C# and Visual C# .....	26
3.2.4. Visual Basic .NET.....	27
3.2.5. Microsoft Small Basic.....	27
3.2.6. Java.....	28
3.3. Summary .....	29
4. Microcomputers and Microcontrollers.....	31
5. Methodology .....	33
5.1. Summary of earlier sections.....	33
5.2. Introduction.....	33
5.3. Research Question.....	34
5.4. Aims and objectives .....	34
5.5. Research involving children.....	34
5.6. Research methods and methodologies .....	36
5.7. This research .....	39

5.7.1. Participants.....	39
5.7.2. Teams.....	39
5.7.3. Content.....	40
5.7.4. Tasks.....	40
5.7.5. Further details.....	40
5.8. Data collection.....	41
5.8.1. Surveys.....	41
5.8.2. Observations.....	41
5.9. Ethics for surveys, observations, interviews and discussions.....	41
6. Case study.....	43
6.1. Introduction.....	43
6.2. Analysis of the event.....	43
6.2.1. Content.....	43
6.2.2. Students.....	44
6.2.3. Surveys.....	45
6.2.4. Discussion.....	45
7. Findings.....	47
7.1. Studying computing.....	47
7.2. Career ambitions.....	52
7.3. Computing skills.....	55
7.4. Rating skills.....	66
7.5. Summary.....	69
7.6. Limitations.....	71
7.7. Future improvements.....	72
7.8 Summary.....	73
8. Conclusions.....	74
9. Recommendations and further work.....	76
10. References.....	78
11. Appendices.....	80
Appendix 1 – Literature review.....	80
I. INTRODUCTION.....	80
II. THE ENROLMENT CRISIS.....	81
III. POSSIBLE REASONS FOR LACK OF INTEREST IN CS.....	82
A. <i>Outsourcing</i> .....	82

B. <i>CS isn't cool</i> .....	82
C. <i>Other reasons</i> .....	83
IV. CS RATHER THAN ICT.....	83
V. GOVERNMENT AND INDUSTRY SUPPORT.....	84
VI. POSSIBLE SOLUTIONS .....	84
A. <i>Tailored courses</i> .....	84
B. <i>Improving and modernizing courses</i> .....	85
C. <i>Focusing courses around a current trend</i> .....	85
D. <i>Make programming more accessible</i> .....	85
E. <i>Different teaching approaches and learning techniques</i> .....	86
F. <i>Outreach projects</i> .....	87
VII. CONCLUSION .....	87
Appendix 2 – Communications with local schools.....	90
Initial email .....	90
Replies received and subsequent conversations.....	90
References.....	97
Appendix 3 – Surveying teachers via an informal discussion .....	98
Initial post .....	98
Replies received .....	98
Appendix 4 – Acquiring knowledge to teach the new National Curriculum.....	101
References.....	101
Appendix 5 – The new National Curriculum.....	102
References.....	102
Appendix 6 – Programming languages/tools/environments for education .....	103
Visual programming languages/tools/environments.....	103
Text-based programming languages .....	113
References.....	123
Appendix 7 – Case Study: Tasks – Further details .....	124
Part 1 .....	124
Part 2.....	124
References.....	125
Appendix 8 – Case Study: Event timetable.....	126
Appendix 9 – Case Study: Tasks Worksheets/Hand-outs.....	127
Scratch introduction .....	127

Part 1 .....	129
Part 2 – Fun with the Raspberry Pi .....	134
Appendix 10 – Case Study: Challenges Worksheets/Hand-outs.....	143
Part 1 challenges .....	143
Part 2 challenges .....	144
Appendix 11 – Case Study: Advice for event staff (people who will help run the event).....	145
Part 1 .....	145
Part 2 .....	148
Appendix 12 – Case Study: Guidance for the event organiser.....	150
Setting up the Raspberry Pi computers .....	150
Introductions to each part.....	150
Task and challenge worksheets .....	151
Resources .....	151
Appendix 13 – Case Study: Survey .....	152
Introduction for the surveys .....	152
Before event survey.....	152
After event survey .....	154
Appendix 14 – Case Study: Observations.....	155
Appendix 15 - Research Information Sheet .....	156
Introduction to the research.....	156
Aims and objectives .....	156
How results/data will be collected .....	157
How the results/data will be used.....	157
Your rights .....	157
Contact .....	157
Appendix 16 – Suitability of the Raspberry Pi .....	158
References.....	158
Appendix 17 – Additional survey results.....	160
Future ambitions: Studying at Bournemouth University .....	160
Other.....	162
Discussion .....	163
Appendix 18 – Learning resources and links .....	164
Professional bodies, working groups and government organisations .....	164
Computing clubs .....	164

Online learning.....	164
Teaching resources.....	165
Robots to teaching computing.....	166
Other.....	166



## Figures

Figure 1: Students accepting a place on a computing course 2004 - 2012.....	18
Figure 2: Students accepting a place on a computing course 2004 – 2012 - by gender.....	18
Figure 3: Students choosing specific computing topics 2004 - 2011.....	18
Figure 4: Male students choosing specific computing topics 2004 - 2011.....	19
Figure 5: Female students choosing specific computing topics 2004 - 2011.....	19
Figure 6: How likely students will choose ICT or Computing as a GCSE option or an equivalent.....	47
Figure 7: How likely students will choose ICT or Computing as a GCSE option or an equivalent - Differences between surveys.....	47
Figure 8: Box plot for the “How likely students will choose ICT or Computing as a GCSE option or an equivalent” question.....	48
Figure 9: How likely students will study AS/A level Computing or a college computing course.....	49
Figure 10: How likely students will study AS/A level Computing or a college computing course - Differences between surveys.....	49
Figure 11: Box plot for the “How likely are you to study AS/A level Computing or a college computing course” question.....	50
Figure 12: How likely students will choose to study a computing course at university.....	51
Figure 13: How likely students will choose to study a computing course at university – Differences between surveys.....	51
Figure 14: Box plot for the “How likely students will choose to study a computing course at university” question.....	51
Figure 15: How likely students think they will get a job in the computing industry.....	52
Figure 16: How likely students think they will get a job in the computing industry – Differences between surveys.....	52
Figure 17: Box plot for the “How likely do you think you will get a job in the computing industry” question.....	53
Figure 18: Students who have considered working in the computing industry after leaving education – Before event.....	54
Figure 19: Students who have considered working in the computing industry after leaving education – After event.....	54
Figure 20: Sectors of the computing industry students are most interested in.....	54
Figure 21: Students motivation/reasons for wanting to work in the computing industry.....	55
Figure 22: Students confidence around describing an ‘if’ statement.....	56
Figure 23: Students confidence around describing an ‘if’ statement – Differences between surveys.....	56
Figure 24: Box plot for the “If I asked you to describe an ‘if’ statement how confident would you be with your reply” question.....	57
Figure 25: Students confidence around describing a loop.....	58
Figure 26: Students confidence around describing a loop – Differences between surveys.....	58

Figure 27: Box plot for the “If I asked you to describe what a loop is how confident would you be with your reply” question.....	59
Figure 28: Students confidence around describing variables .....	59
Figure 29: Students confidence around describing variables – Differences between surveys ..	59
Figure 30: Box plot for “If I asked you to describe what a variable is how confident would you be with your reply” question.....	60
Figure 31: Students confidence around programming with Scratch.....	61
Figure 32: Students confidence around programming with Scratch – Differences between surveys .....	61
Figure 33: Box plot for the “How confident do you feel about using Scratch to program” question.....	62
Figure 34: Students confidence learning new programming languages .....	63
Figure 35: Students confidence learning new programming languages – Differences between surveys .....	63
Figure 36: Box plot for the “How confident do you feel about learning new programming languages” question .....	64
Figure 37: Students confidence using any programming language .....	64
Figure 38: Students confidence using any programming language – Differences between surveys .....	64
Figure 39: Box plot for the “How confident do you feel about using any programming language” question.....	65
Figure 40: Students rating of their computing skills.....	66
Figure 41: Students rating of their computing skills – Differences between surveys .....	66
Figure 42: Box plot for the “How would you rate your computing skills” question.....	67
Figure 43: Students rating of their programming skills .....	67
Figure 44: Students rating of their programming skills – Differences between surveys.....	67
Figure 45: Box plot for the “How would you rate your programming skills” question .....	68

## Tables

Table 1: Weighted scores for all programming languages/tools/environments considered .....	29
Table 2: Statistics for programming languages/tools considered in regards to teaching.....	30
Table 3: Quartiles for the “How likely students will choose ICT or Computing as a GCSE option or an equivalent” question .....	48
Table 4: Averages for the “How likely students will choose ICT or Computing as a GCSE option or an equivalent” question .....	48
Table 5: Quartiles for the “How likely are you to study AS/A level Computing or a college computing course” question.....	50
Table 6: Averages for the “How likely are you to study AS/A level Computing or a college computing course” question.....	50
Table 7: Quartiles for the “How likely students will choose to study a computing course at university” question .....	51
Table 8: Averages for the “How likely students will choose to study a computing course at university” question .....	51
Table 9: Quartiles for the “How likely do you think you will get a job in the computing industry” question .....	53
Table 10: Averages for the “How likely do you think you will get a job in the computing industry” question .....	53
Table 11: Quartiles for the “If I asked you to describe an ‘if’ statement how confident would you be with your reply” question .....	57
Table 12: Averages for the “If I asked you to describe an ‘if’ statement how confident would you be with your reply” question.....	57
Table 13: Quartiles for the “If I asked you to describe what a loop is how confident would you be with your reply” question.....	58
Table 14: Averages for the “If I asked you to describe what a loop is how confident would you be with your reply” question.....	58
Table 15: Quartiles for the “If I asked you to describe what a variable is how confident would you be with your reply” question.....	60
Table 16: Averages for the “If I asked you to describe what a variable is how confident would you be with your reply” question.....	60
Table 17: Quartiles for the “How confident do you feel about using Scratch to program” question.....	62
Table 18: Averages for the “How confident do you feel about using Scratch to program” question.....	62
Table 19: Quartiles for the “How confident do you feel about learning new programming languages” question .....	64
Table 20: Averages for the “How confident do you feel about learning new programming languages” question .....	64
Table 21: Quartiles for the “How confident do you feel about using any programming language” question.....	65

Table 22: Averages for the “How confident do you feel about using any programming language” question.....	65
Table 23: Quartiles for the “How would you rate your computing skills” question .....	67
Table 24: Averages for the “How would you rate your computing skills” question.....	67
Table 25: Quartiles for the “How would you rate your programming skills” question.....	68
Table 26: Averages for the “How would you rate your programming skills” question .....	68

## Glossary

BCS = BCS, The Chartered Institute for IT - <http://bcs.org>

BU = Bournemouth University – <http://bournemouth.ac.uk>

CAS = Computing at Schools Working Group - <http://computingatschool.org.uk>

CPD = Continuous Professional Development

CPU = Central Processing Unit

CRA = The Computing Research Association - <http://cra.org>

CS = Computer Science

DfE = Department for Education (UK) - <http://www.gov.uk/dfes>

EdD = Doctorate in Education

FAQ = Frequently Asked Questions

GPIO = General Purpose Input/Output

HESA = Higher Education Statistics Agency - <http://www.hesa.ac.uk/>

ICT = Information and Communications Technology

IDE = Integrated Development Environment

KS = Key Stage

KS1 = Key Stage 1 (5 to 7 year olds - years 1 and 2)

KS2 = Key Stage 2 (7 to 11 year olds - years 3 to 6)

KS3 = Key Stage 3 (11 to 14 year olds - years 7 to 9)

KS4 = Key Stage 4 (14 to 16 year olds - years 10 and 11)

KVM = Keyboard, Video and Mouse

MIT = Massachusetts Institute of Technology

NC = National Curriculum

NCB = National Children's Bureau - <http://www.ncb.org.uk>

OOP = Object-Oriented Programming

PhD = Doctorate in Philosophy

RAM = Random Access Memory

UCAS = Universities and Colleges Admissions Service - <http://ucas.com>

Year 1 = 5 to 6 year olds (KS1)

Year 2 = 6 to 7 year olds (KS1)

Year 3 = 7 to 8 year olds (KS2)

Year 4 = 8 to 9 year olds (KS2)

Year 5 = 9 to 10 year olds (KS2)

Year 6 = 10 to 11 year olds (KS2)

Year 7 = 11 to 12 year olds (KS3)

Year 8 = 12 to 13 year olds (KS3)

Year 9 = 13 to 14 year olds (KS3)

Year 10 = 14 to 15 year olds (KS4)

Year 11 = 15 to 16 year olds (KS4)

## 1. Introduction

Computing has evolved at a remarkable pace in recent years and is becoming ubiquitous and essential for modern life; however computing education hasn't evolved as quickly. This has created a skills gap with Further and Higher Education institutions having difficulty finding suitable students and likewise industry has problems finding suitable employees.

Studies have shown that interest in studying and pursuing computing careers is low despite an ever increasing demand for computing skills (Cooper et al. 2010; Morelli et al. 2010; Purewal Jr. 2010). This applies to all businesses (not just the computing industry) due to the prevalence of computing in modern society. In addition computing skills are highly valued as they demonstrate other skills such as problem solving, design, creativity and logic skills.

Possible reasons for this lack of interest have been identified as: misconceptions of what computing education and careers involve, the way computing is taught with failures to show the relevance of computing, outdated content, lack of computer science content, and so forth (Albinson 2013).

This dissertation will look at the reasons behind this observed lack of interest in computing and ways to make computing more appealing. It will focus around a case study of a university outreach computing event for secondary schools designed to enhance students' perceptions and understanding of computing. It will also provide teachers with a Continuing Professional Development (CPD) opportunity to learn more about computing and provide ideas on interactive teaching methods. It will include the use of physical computing such as showing the hardware which makes computers work and using electronics with computers as inputs and outputs of a program. It is designed to make programming more fun and engaging by showing the effects of programming over a physical object such as turning on a light and how inputs such as switches can be used.

The next section details the background to the project along with related literature. Section 3 compares programming languages/tools/environments used in education. Section 4 reviews microcomputers and microcontrollers which can be used to introduce physical computing into education. Section 5 contains the research methodology, followed by section 6 describing the case study. Section 7 covers the findings from the research. The dissertation then finishes with conclusions, recommendations and further work sections.

## 2. Background to the project

In preparation for this dissertation the researcher conducted a literature review (Albinson 2013) which reviews the factors affecting the decline in undergraduate university Computer Science (CS) course enrolments and approaches for solving this problem (see appendix 1). This showed that students' impressions of computing courses in Further and Higher Education and computing careers is low despite its importance in modern society. There is an observed reduction in university CS course enrolments from around 2000 with a slight increase in recent years as various approaches are used to try and reverse the trend<sup>1</sup>. There are a variety of reasons for this unpopularity such as: misconceptions of what computing education and careers involve and that outsourcing has reduced job availability, poor quality computing education in schools, outdated university courses and so forth.

As a response, strategies are being employed to improve students' perceptions and understanding of computing and increase their computing knowledge including: improving school curriculums and guidelines to make computing more prominent and to cover more computing content, improving and modernising computing courses, tailoring introductory courses to particular interests and careers, focussing courses around a current trend, making programming more accessible, using different teaching approaches and learning techniques, outreach projects and so forth. Evidence shows that these approaches improve students' perceptions and understanding of computing and consequently university enrolments increase (Albinson 2013).

### 2.1. English university enrolments

Unlike the USA which has the Taulbee Survey (CRA 2013) the UK does not have a definitive set of figures on university computing course enrolments and retention. However UCAS (2013a) publishes figures on university applications including offers made and their acceptance (UCAS 2013b)<sup>2</sup>. These figures include courses chosen by subject/topic and subject groupings<sup>3</sup> and there are different versions for the whole UK and its individual countries. Prior to 2012 there was no subject group specifically for computing and computing subjects were primarily in the Mathematical & Computer Sciences group. Computing subjects were Computer Science, Information Systems, Software Engineering and Artificial

---

<sup>1</sup> The data used in the literature review was from the USA due to greater availability of papers and statistics on computing education and university computing course enrolments than the UK.

<sup>2</sup> More useful data can be found on the UCAS website (UCAS 2013c)

<sup>3</sup> They use the Joint Academic Coding System for this.



Intelligence<sup>4</sup>. From 2012 they use an updated subject list (HESA 2013) which includes a Computer Science group for all computing courses. Therefore using the groups and subjects identified as computing we can get a reasonably accurate idea of computing enrolments<sup>5</sup>. The change in grouping subjects is likely to make the results for 2012 different to previous years due to a larger amount of subjects specified as computing but it should more accurately identify computing courses.

Acceptance rates<sup>6</sup> for computing courses in England<sup>7</sup> (Figure 1) show reductions from 2004 to 2007 and significant increases to 2009 before smaller increases. Despite the extra courses being identified as computing in 2012 there is only a moderate increase<sup>8</sup>. These results are similar to American universities which also saw decreases until around 2007 before steady increases (Albinson 2013).

Splitting the acceptance rate by gender (Figure 2) the results for males (who are in the majority) are similar to the overall results (Figure 1), whereas the acceptance rate for females is much smaller with very small variations per year. This also applies to individual topics<sup>9</sup> (Figure 3); there are similar patterns for Computer Science and Information Systems while Software Engineering is steadily increasing and Artificial Intelligence has consistently low figures<sup>10</sup>.

---

<sup>4</sup> However some computing courses were probably within the “Mathematical & Computer Sciences: any area” or the “Others in Mathematical & Computer Sciences” subjects but it is not possible to differentiate which are computing courses within these subjects so they cannot be included in computing course totals.

<sup>5</sup> However there may be courses in other groups/subjects which have computing content as part of their course such as Business and IT which wouldn’t specifically be considered as a computing course. Also it is not an ideal or particularly accurate set of statistics but in the absence of a dedicated survey of UK University computing courses it appears to be the best available data.

<sup>6</sup> The acceptance figures are categorized according to the subject of courses which students accepted and thus we can assume they enrolled on these courses/subjects.

<sup>7</sup> England is used instead of the whole of the UK as a) the case study focuses on English schools and b) there are different funding policies within the countries in the UK which are likely to affect application figures.

<sup>8</sup> Thus it appears the new Computer Science group contains a similar amount of courses than were identified as computing in the previous years. However there is only 1 year of statistics using this new group and it will require more years of statistics to fully analyse the new grouping’s effect.

<sup>9</sup> 2012 results are excluded as they use different groupings and subjects and as there is only 1 year of results using these it is difficult to compare them to the 2004-2011 results.

<sup>10</sup> This could be due to little interest in the subject or limited course availability.

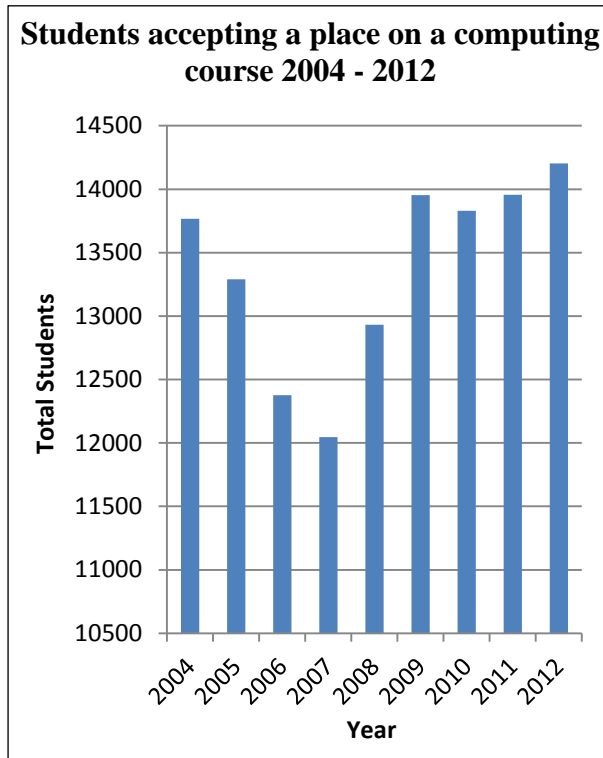


Figure 1: Students accepting a place on a computing course 2004 - 2012

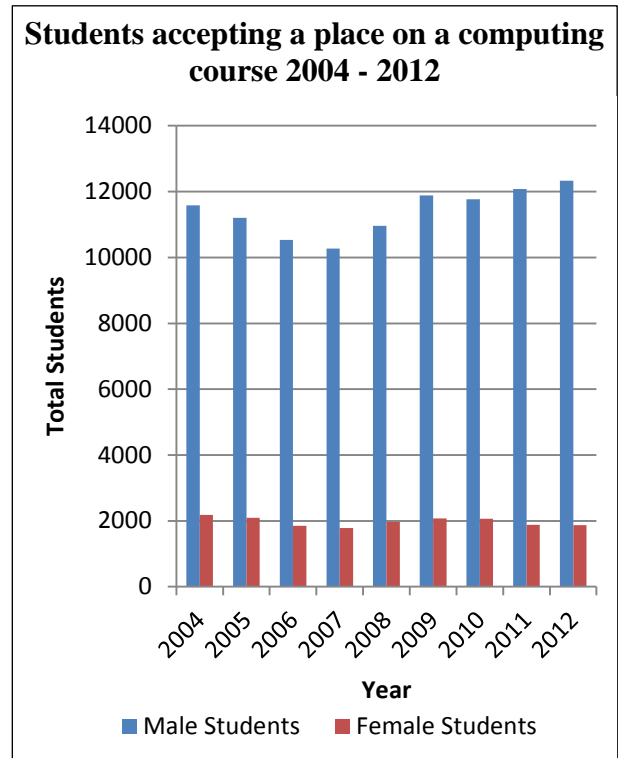


Figure 2: Students accepting a place on a computing course 2004 – 2012 - by gender

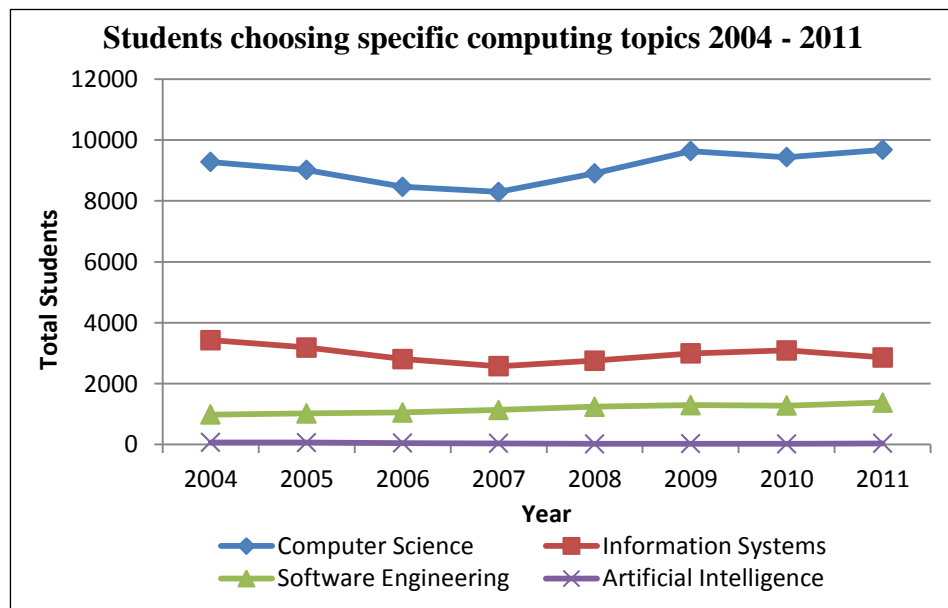


Figure 3: Students choosing specific computing topics 2004 - 2011

Splitting topic choices by gender (Figure 4 and Figure 5) shows that females have similar results to males but Information Systems is more popular.

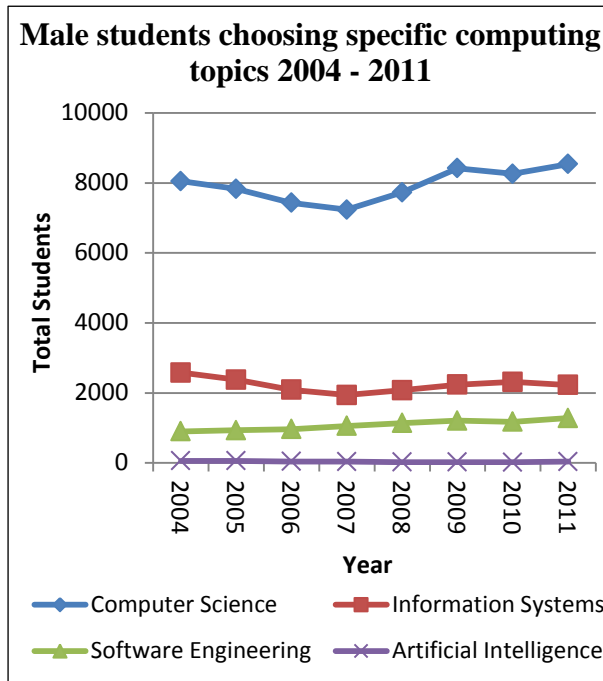


Figure 4: Male students choosing specific computing topics 2004 - 2011

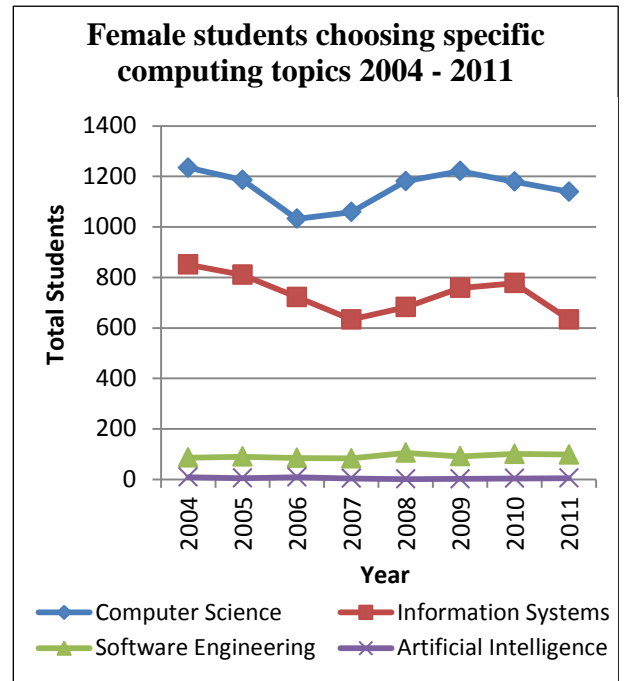


Figure 5: Female students choosing specific computing topics 2004 - 2011

## 2.2. Computing in schools

Albinson (2013) shows that the computing experiences, especially teaching, that school students' receive influences their opinions of computing and considerations towards future computing study and careers. Therefore an informal investigation was carried out to examine the computing teaching offered in English schools and if teachers are prepared for the reformed/new National Curriculum (NC) with its enhanced computing content (see 2.3.) and whether assistance with it would be useful. This was done via discussions with a sample of secondary schools in the local area (see appendix 2) and via an informal discussion (see appendix 3). The replies showed that some schools have teachers with CS degrees and are able to easily adjust to using the new NC. However many schools have non-CS staff and the new NC introduces plenty of new content which the average computing teacher will not have covered before and will need training to understand it (either self-taught or formal education). Teachers are understandably concerned about how to teach the new content especially as there is little government advice on how to interpret and teach the new NC; see appendix 4 for more information. This situation shows a need for resources and advice to be made available to not only assist in the transition to the new NC but to enhance the quality of computing teaching.

### 2.3. The new National Curriculum

This need to improve computing education in schools has been recognised by the UK government and a new Computing subject to replace ICT has been included in the proposed new National Curriculum for England (due to come into effect September 2014). Current computing education (ICT) commonly only covers Digital Literacy (how to use basic applications such as word processors and the internet) because:

- a) ICT in the NC is flexible and low level/simple content can be chosen to allow non-specialists to teach it,
- b) many teachers only have digital literacy skills,
- c) a lack of professional development among computing teachers,
- d) school infrastructures can restrict teaching more advanced computing content

(The Royal Society 2012). The proposed Computing subject covers a much larger/complete range of content and defines Computing as Computer Science, Digital Literacy and IT (DfE 2013a). See appendix 5 for further details on the new NC.

Informal discussions show that there is some confusion among teachers on how to interpret and teach the new NC. For example the requirement for Key Stage 1 students to be able to write and test simple programs could be difficult with children of this age as they may not have fully developed reading skills and as such reading code would be especially difficult. Visual programming tools designed for teaching, such as Scratch, reduce complexity but as it uses blocks instead of text-based code it may not meet the requirements for writing code. Even these tools could be too complex for young children and in response to this a simplified junior version of Scratch (ScratchJr) is being developed for 3-8 year olds (ScratchJr 2013).

There is also confusion on what content should be covered to meet the aims of the curriculum; teachers need to know what is vital to cover and what can be removed to ensure they can fit the most relevant content into the available lesson time. The curriculum is intentionally short with summarised content as aims/objectives to allow for flexibility when delivering the content. While this is useful to allow teachers to adapt content to meet students' needs, school requirements, facilities etc. many teachers would benefit from more in-depth guidelines especially those with limited computing skills (for example see the views in appendices 2 and 3).

## 2.4. Implementing the new National Curriculum

With the new Computing subject in the new NC set to come into effect in 2014 planning is required to enable a smooth transition and implementation. There are two trains of thought on this; a) ignore the new content for now and wait until planning for 2014-15 begins or b) start planning now and introduce some new content for a phased implementation. Whereas it may appear a little early to start using the new NC's content<sup>11</sup>, implementing it into planning as soon as possible allows for a more phased implementation and evaluation before it becomes a requirement in 2014.

Coincidentally phased implementation of the new NC was considered but it was decided, given the importance of providing the benefits it offers, it was best not to delay it and to introduce it all in September 2014 (DfE 2013b, p.15).

Teachers most popular implementation is a phased approach, choosing to include some new computing content from September 2013 (as seen in the views in appendices 2 and 3), realising it is impractical to introduce the entire new computing NC straight away. Existing students will not have prior knowledge of the Computing subject in the new NC and therefore meeting its requirements could be difficult. Introducing as much new computing content as feasible from September 2013 while retaining some existing ICT content will prepare students for the new NC from September 2014 while not being too complex or daunting. However even with this approach it may take many years for students' computing skills to increase to the required levels to fully embrace the new NC's content; compare the skills and advantages students learning the computing content of the new NC from the start of school will have by the time they reach secondary school to those in secondary school now who were not taught the new Computing subject/curriculum<sup>12</sup>. Additionally the range of skills of the students could vary dramatically, for example each feeder school to a secondary school may teach computing differently; some may cover a lot of content, some very little, and students' skills vary accordingly.

---

<sup>11</sup> Especially as it isn't fully approved (correct at time of writing)

<sup>12</sup> These students would have been taught ICT which could have just contained basic content such as how to use certain programs like office applications and may not have covered content such as programming, logic etc.

Until the effect of the improved curriculum is felt, when all students entering secondary school should have the same understanding of computing, schools should make their content flexible. This flexibility would still be useful when the new NC is fully implemented<sup>13</sup> as some students may have difficulty with it and have poor skills due to lack of interest, learning difficulties etc.

## 2.5. Teaching programming

Programming is useful for introducing computing principles/fundamentals. As syntax and concepts are similar between programming languages then if programming concepts are properly introduced students should be able to easily transition to using other languages. Converting to Computer Science (2013) discusses whether teachers (and the same could be said for students) should learn programming by learning programming concepts rather than a specific programming language. This approach will fully introduce programming concepts, principles/fundamentals, etc., and enable easy transition between programming languages. However it may be a too difficult introduction for teachers/students with limited computing skills and perhaps learning only one language would be a more suitable introduction.

Early year students may have poor typing skills so would struggle with text-based languages and therefore arguments are made for using visual non-text based programming languages/tools to make programming easy to understand.

Informal discussions recommend using graphical outputs to provide visual feedback as more motivating results from programming. There are also suggestions to progress onto text-based languages for added complexity and to introduce debugging skills, more relevance to languages used in industry such as Java etc. The new NC specifies that from Key Stage 3 students must use two or more programming languages of which one must be text-based (DfE 2013a). It is also advisable to teach debugging skills such as producing basic rules/tips for identifying a problem before asking the teacher for help<sup>14</sup>; another approach is purposely providing students with faulty code for them to try and debug.

---

<sup>13</sup> When students should have all covered the same content and within theory have the same skills/level of understanding.

<sup>14</sup> For example PythonCode.co.uk (2013) contains debug rules; it is based around Python but in general can apply to most languages.

Varying programming languages used throughout education helps students understand programming concepts and see their relevance and reduces the chance of them getting bored “we used Scratch last year and the year before that”. Also it helps build up complexity and as many languages have similar syntax the students will hopefully see the similarities and have more confidence to try other programming languages.

## 2.6. Teaching computer concepts

While teaching programming is valuable and can be used to introduce computing principles/fundamentals, debugging skills etc. it is vital to ensure other areas are not neglected. The new NC specifies a range of topics in the areas of IT, Digital Literacy and Computer Science. Teachers must cover a broad range of content including the fundamental principles of Computer Science, computational thinking, how systems work (both hardware and software including communication such as the use of networks), logic, problem solving and algorithmic thinking (DfE 2013a). Many of these skills can be introduced via programming but content should ideally be taught via multiple approaches. Grover (2013) discusses how learning to code is not enough and CS concepts/fundamentals (including programming fundamentals) and computational thinking skills should be taught as they are essential skills.

Students who learn programming without knowing these skills struggle to fully understand the purpose of key concepts such as Booleans, conditions, loops etc. within programs. Additionally programming teaching tools like Scratch, App Inventor and Alice and some introductory programming courses can fail to teach programming fundamentals. Their focus on quickly and easily creating programs, while very motivational, can leave students unable to read/understand unfamiliar code (especially text-based programming) and debug problems<sup>15</sup>. Similarly Wing (2006) explains how computational thinking should be a core skill for students in addition to reading, writing and arithmetic.

---

<sup>15</sup> They may have just copied code, or blocks representing programming components/code, to complete a task without considering what the code does. The simplicity of these tools while a major strength is also a weakness as it makes it too easy to just recreate a tutorial example without considering the code and concepts behind it.

### 3. Comparing programming languages/tools/environments

There are various programming languages/tools/environments used in education with varying differences, resources, tutorials and other reasons for choosing them. Some are visual programming tools/environments and are primarily designed for making programming simple and easy to learn so offer an excellent introduction to programming. Others are more traditional text-based languages/tools/environments which are either: used in industry, or are similar to languages used in industry and aim to be simpler to understand and learn programming with but may be more difficult than visual programming languages/tools/environments.

A feature analysis (Pfleeger 2001, p.509) will be conducted for comparing each language/tool/environment considered. Various attributes have been chosen to consider the value of the language/tool/environment with scores out of 5 on how satisfactorily it meets these key criteria, where 1 is completely unsatisfactory and 5 is completely satisfactory. Each attribute also has a weight/level of importance which is also out of 5. The weights are personal reasoning and are based on importance for teaching with key areas identified as: relevance to education and how it can assist with teaching/learning, usability, and ability to inspire and motivate students to program. However other people may have different views on the importance of these attributes and would give them different values. This would therefore affect the scores and perhaps even show different languages are more useful than the ones identified in this study. The attributes that have been selected and their weights are:

- **Usability of the programming tool/environment** – How intuitive and easy to use is: the environment, its features etc.? **Weight: 4**
- **Ease of use and intuitiveness of the programming language** – How complex is creating a program? **Weight: 4**
- **Programming concepts covered** – Does it cover a full range of programming concepts including advanced programming concepts such as Object-Oriented Programming (OOP)? **Weight: 3**
- **Relevance to industry** – Is it used in industry or has a similar approach to an industry tool? **Weight: 2**
- **Ability to create real-world and relevant applications** – Are the applications created relatable to the real-world? **Weight: 3**
- **Interactive features** – How interactive is it? **Weight: 2**



- **Motivational potential** – Will it motivate students? **Weight: 3**
- **Quality of documentation and amount of teaching resources available. Weight: 4**
- **Longevity and update frequency** – Is the tool/environment frequently updated to remove any problems, enhance the system etc. Is there support for keeping it available?<sup>16</sup> **Weight: 2**

**Potential total: 45**

**Potential weighted total: 135**

Various popular programming languages/tools/environments for education are briefly introduced below and are explained in more detail in appendix 6 along with full details of the feature analysis.

### **3.1. Visual programming languages/tools/environments**

Visual programming languages/tools/environments (e.g. App Inventor, Scratch and Alice) are designed to make programming easier by removing the need to understand specific syntax. Instead they work by having components for programming elements (loops, variables etc.) which can be dragged into the tool/environment to build up the program and only fit together if semantically correct. This approach allows students to focus on understanding programming concepts rather than having to use complex and potentially confusing syntax.

### **3.2. Text-based programming languages**

Although text-based programming languages are more complicated than visual programming languages/tools they are more common in industry and can teach more advanced skills including debugging skills.

#### **3.2.1. Python**

Python is probably the most common text-based language used in schools based on informal discussions. It could be argued given its dominance that it should be the only text-based language used at KS3 to introduce consistency, provide transferrable skills, and to focus on one language not many.

---

<sup>16</sup> For example if it isn't updated often it may show there is little interest in its longevity and development could stop at any point and it may even become unavailable.

### 3.2.2. Logo

Logo (Logic Oriented & Graphic Oriented programming language) is a very basic language designed for teaching programming to young children<sup>17</sup>. Its most popular feature is drawing with either a physical turtle robot (similar to Roamer (2013)) or a turtle on a computer screen. It also includes support for other programming concepts such as loops/repeat, functions, lists and arrays. There is no specific official Logo tool or language and there are many implementations and dialects but they all share the same philosophy and similar syntax.

Logo's simple commands and philosophy make it ideal for young children; for example drawing a square is as easy as:

```
FORWARD 150  
LEFT 90  
FORWARD 150  
LEFT 90  
FORWARD 150  
LEFT 90  
FORWARD 150  
LEFT 90
```

This is easy to interpret as it is simply directions, distances and angles.

To simplify this further a repeat can be introduced and the code becomes:

```
REPEAT 4 [FORWARD 150 LEFT 90]
```

This can also be easily interpreted; repeat the contents of the brackets 4 times (forward 150 and rotate 90 degrees left).

### 3.2.3. C# and Visual C#

C# is an Object-Oriented Programming (OOP) language based on C and is type-safe and designed to be simple and modern (Microsoft 2012a). It can create either command line/console applications (native C#) or Windows applications with a Graphical User Interface (GUI) (via Visual C#). The use of Visual C# allows for Rapid Application Development (RAD) as GUI elements can be easily created by dragging or drawing them onto the IDEs form/window designer. Code can then be added to them e.g. call a function after clicking a button. Although it is designed to be easy, there is extra code to understand compared to some languages due to the use of OOP. It has many predefined libraries to handle complex functions and is useful for teaching/learning all programming concepts in an OOP way. Also due to its popularity there are many teaching/learning resources available for it.

---

<sup>17</sup> Although it could be used to provide a basic introduction to programming for any age

### 3.2.4. Visual Basic .NET

Visual Basic (VB) .NET is similar to C#/Visual C# as they both use the Visual Studio IDE, are object-oriented and run on the .NET framework. VB .NET is a complete programming language whereas Visual C# integrates the C# language into the Visual Studio IDE and adds the ability to create GUIs. VB .NET is primarily focused on creating GUI Windows applications but can also create console applications. Both VB .NET and C#/Visual C# are popular and reasonably easy to use and have similar amounts of learning resources available for them; however with C#/Visual C# being based on the well-established C programming language (and therefore sharing similar syntax with other languages) and being designed to be simple and modern it is probably a better choice.

### 3.2.5. Microsoft Small Basic

Microsoft Small Basic (Microsoft 2013a) is based on Basic, from which VB and VB .NET originate, and is simplified with a much smaller syntax containing only 14 keywords<sup>18</sup>. It is designed for 10 to 16 year olds but is useful as a first language for any age. Despite its simplicity it has a rich programming environment and set of libraries to allow beginners to easily create significant programs (Microsoft 2013b). The simple syntax makes commands easy to understand and it is also similar to standard programming languages like C#. An intellisense list is shown as the user types to assist them with the code they wish to write and descriptions are shown for each word typed/chosen (functions, properties etc.). Although it is basic it covers almost all programming concepts and allows the user to easily and quickly create substantial programs (either command line/console or GUI applications). There is a long introductory document (Microsoft 2012c) which covers all its features via many interesting and well explained examples. There is also: documentation explaining the language, a curriculum (Microsoft 2013d), eBooks (Microsoft 2013e), a user community for sharing projects, and a forum.

---

<sup>18</sup> The keywords are listed and explained on (Microsoft 2012b)

### 3.2.6. Java

Java (2013) is an extremely popular OOP language which is used to create applications to run on almost any device. Its use of the Java Virtual Machine (JVM) enables programmers to only need to write code once and it will work on any JVM-enabled device<sup>19</sup>. It is very popular in industry and with university teaching and to a lesser extent School and Further Education. It is derived from C and C++<sup>20</sup> and has similar syntax but due to its OOP nature and extensive functionality it can be difficult for novices to understand; plus it wasn't designed as a teaching/beginners language. There is extensive documentation, tutorials and examples on the Java website and due to its popularity there are many other resources available. There are tools designed for providing introductions to Java such as Alice (2013), BlueJ (2013) and Greenfoot (2013) which are commonly used to assist with teaching.

---

<sup>19</sup> The process is known as "Write once, run anywhere" (WORA).

<sup>20</sup> Another C derivative

### 3.3. Summary

Table 1 contains the weighted attribute scores for all programming languages/tools/environments considered.

Table 1: Weighted scores for all programming languages/tools/environments considered

<b>Attribute \ Language/Tool/Environment</b>	<b>App Inventor</b>	<b>Scratch</b>	<b>Alice</b>	<b>Python</b>	<b>Logo</b>	<b>C# / Visual C#</b>	<b>Visual Basic .NET</b>	<b>Microsoft Small Basic</b>	<b>Java</b>
<b>Usability of the programming tool/environment</b>	14	16	12	10	18	19	19	16	Not applicable
<b>Ease of use and intuitiveness of the programming language</b>	16	18	10	14	18	17	16	18	12
<b>Programming concepts covered</b>	10.5	9	13.5	12	7.5	13.5	13.5	12	14.25
<b>Relevance to industry</b>	7	4	8	9	2	9	7.5	4	9.5
<b>Ability to create real-world and relevant applications</b>	12	6	6	13.5	3	13.5	13.5	3	14.25
<b>Interactive features</b>	9	7	4	8	4	9	8	2	9.5
<b>Motivational potential</b>	13.5	9	9	12	7.5	13.5	13.5	9	7.5
<b>Quality of documentation and amount of teaching resources available</b>	16	18	12	16	12	18	18	16	18
<b>Longevity and update frequency</b>	9	9	9	8	5	9	9	4	10
<b>Total (Out of 135)</b>	107	96	83.5	102.5	77	121.5	118	84	95 <sup>21</sup>

<sup>21</sup> Note: For Java the potential weighted total is 115 not the usual 135 due to the usability attribute not being applicable.

Table 2 contains the statistics for the programming languages/tools considered.

Table 2: Statistics for programming languages/tools considered in regards to teaching<sup>22</sup>

<b>Name</b>	<b>Type</b>	<b>Weighted total</b>	<b>Weighted average</b>
App Inventor	Visual	107 (79.26%)	3.96
Scratch	Visual	96 (71.11%)	3.56
Alice	Visual	83.5 (61.85%)	3.09
Python	Text-based	102.5 (75.93%)	3.80
Logo	Text-based	77 (57.04%)	2.85
C#/Visual C#	Text-based	121.5 (90%)	4.5
Visual Basic .NET	Text-based	118 (87.41%)	4.37
Microsoft Small Basic	Text-based	84 (62.22%)	3.11
Java	Text-based	95 (82.61%)	4.13

Table 2 appears to show that C#/Visual C#, Visual Basic .NET and Java are the most suitable languages/tools/environments for teaching programming in education (with weighted averages of 4.5, 4.37 and 4.13 respectively) and that visual languages/tools/environments are less useful than text-based languages/tools/environments. However it should be noted that the attributes, reasoning, scores and weights consider education in general, such as concepts they cover, and do not consider specific age groups or different levels or types of course. Therefore this table gives an indication of the overall usefulness of languages and helps establish which is most suitable to use, but the decision should be influenced by the course requirements and content (introductory or advanced content), students' age range and so forth.

---

<sup>22</sup> Note: For Java the potential weighted total is 115 not the usual 135 due to the usability attribute not being applicable. Therefore considering languages should be done via percentages and averages.

## 4. Microcomputers and Microcontrollers

Microcomputers such as the Raspberry Pi (2013) and the BeagleBone (BeagleBoard 2013) and microcontrollers such as the Arduino (2013) can be used with programming to control electronic components (for example a LED or a switch) as inputs and outputs of a program. This can be used to show the hardware which makes computers work and how programming can control electronics. It can also help make programming more fun and engaging by showing the effects of code over physical objects.

Despite having similar components (processors, memory, GPIO<sup>23</sup> pins etc.) the devices are significantly different. The main difference being microcontrollers do not have an operating system and are only able to run code that is added onto it via a computer. Whereas microcomputers have an operating system and are a full computer capable of running much more significant processes, for example graphics processing, multitasking, programming (including creating programs to run on it) and so forth. Microcontrollers typically have significantly slower processors and less memory but due to their simplicity (they just execute one piece of code) they do not require large resources. They can also be smaller and due to their simplicity have significantly lower power consumption. They are both capable of controlling electronics and the device used typically depends on user requirements (power usage versus functionality for example).

The Arduino is probably the most popular microcontroller and has a variety of versions to meet different needs (prices start around £19) and many easy to use electronic components are available for it. There is also a large user community to offer advice, tutorials and so forth.

The Raspberry Pi is probably the most well-known microcomputer and also has a large user community. It has a considerably faster processor and larger memory compared to the Arduino<sup>24</sup> and has excellent capabilities such as superb graphics capable of a full HD output. At approximately £30 it is very affordable and can be used as a computer as well as an electronics controller which is perfect for education and keeping costs low<sup>25</sup>.

---

<sup>23</sup> General Purpose Input/Output

<sup>24</sup> The Raspberry Pi has a 700MHz processor and up to 512MB RAM whereas the Arduino only has a 16MHz processor and 2KB of RAM. However the simplicity of the Arduino reduces the need for fast processors and large amounts of memory.

<sup>25</sup> For example programs can be written on the Raspberry Pi and run on it to control electronic components via its GPIO ports. Of course programs can also be written on another computer and then transferred onto the Raspberry Pi to run there.

The BeagleBone is similar to the Raspberry Pi with comparable specifications but is more expensive and has less graphics power and outputs. However it has more GPIO pins and support for electronic components. A larger and more powerful version (the BeagleBoard) is available which would allow for BeagleBone projects to be scaled up (Maker Media 2013).

The device chosen depends on requirements, projects it will be used with, whether computing features are required, the connections required and so forth<sup>26</sup>.

---

<sup>26</sup> The devices are compared in more detail at (Maker Media 2013). Note: This is a little out of date and shows the Raspberry Pi having 256MB of RAM which has since been improved to 512MB.



## **5. Methodology**

### **5.1. Summary of earlier sections**

Previous sections have explained the current state of computing in general and how there is a lack of interest in computing careers and Further and Higher Education courses. Section [2](#) showed the current state of computing education including how computing is taught now and problems with it such as the need for improvement, lack of computing skills among teachers, and planned future improvements. Programming languages/tools/resources, microcontrollers and microcomputers have been evaluated (Sections [3](#) and [4](#)) as well as discussing a variety of ways to improve computing education (Section [2](#)). This knowledge will now be used as a basis of a project to investigate if computing is improved to be more fun, motivational, and relevant via interactive teaching methods can it enhance perceptions and understanding of computing in secondary schools?

### **5.2. Introduction**

It is evident from the literature that there are numerous issues involving the teaching of computing. A good starting point for any research would be to find out from local school staff and students their attitudes towards computing.

Due to the identified need to improve students' perceptions of computing an outreach event will be created. It will be designed to make computing fun, motivational, relevant and have real-world applications. Staff also expressed interest in increasing physical computing in schools such as showing the hardware which makes computers work and using electronics with computers as inputs and outputs of a program as a result of coding such as controlling lights, sensors, motors and so forth. The event will be designed to make programming more fun and engaging by showing the effects of programming over a physical object such as turning on a light and how inputs such as switches can be used.

A university outreach project for secondary schools provides a perfect opportunity to investigate whether interactive programming tasks with physical outputs can improve motivation to learn programming and consequently computing.

### 5.3. Research Question

Can interactive teaching methods enhance students' perceptions and understanding of computing and increase their computing knowledge?

### 5.4. Aims and objectives

The project will be designed to meet the following aims and objectives, and to answer the research question.

#### **Aims:**

- To enhance students' perceptions and understanding of computing via an outreach computing event
- To provide teachers with a Continuing Professional Development (CPD) opportunity to learn more about computing and provide ideas on interactive teaching methods

#### **Objectives:**

- To provide fun and motivational programming examples which demonstrate fundamental programming concepts, physical computing and programming with electronics
- To show the relevance of computing via hands-on examples ideally with as many real-world and relevant examples as possible
- To provide teachers and students with a CPD opportunity to learn more about computing and to provide teachers with ideas for activities they can run with their students (perhaps continuing on from the events activities or repeating them with new students)
- To observe measured improvement in students' perceptions and understanding of computing

### 5.5. Research involving children

It is important that we first consider which research methods are feasible for using with the children involved. Care needs to be taken when planning research involving children as due to their age they may not have the maturity and cognitive skills to understand certain research methods and/or what is being asked. Depending on their age certain questioning techniques and research methods can be understood.

De Leeuw (2011) explains, and The National Children's Bureau "NCB" (2011, p.17) agrees (however with slightly different ages ranges), how children from the ages of 7 can be surveyed as this is a major developmental milestone<sup>27</sup> and they should have the required cognitive skills and maturity to understand surveys and complete self-reports<sup>28</sup>. As children between the ages of 7-18 are still developing their thinking, logic, reasoning, memory, language and social skills the complexity of research needs to be adjusted dependent on age<sup>29</sup>. Children's understanding of surveys and the reliability of their answers also increase as a result of developing these skills.

7-12 year olds have sufficient language skills for individual semi-structured interviews with structured interviews being feasible from around age 9. However as reading skills are still developing simple language should be used<sup>30</sup> along with checks to ensure the group understand words used. Question complexity and the number of response categories require consideration as their memory capacity, memory speed and cognitive abilities are still developing. Also they may not have emotional or social skills to give reliable answers; they are easily influenced, give insincere answers or answers they think are desired or popular.

12-16 year olds<sup>31</sup> have well developed cognitive skills and understand logical operators and negations. Therefore questions designed for adults can be used as long as they are carefully worded to avoid ambiguity. Their memory speed is not fully developed (although their memory capacity is) so will still require a reasonable amount of time to complete the questions. Peer pressure can be a problem with reliability of results as children in this age group are commonly influenced by their peers<sup>32</sup>.

Children 16-18 can be regarded as adults as they have fully developed cognitive and information processing skills however they do not have fully developed social or organisational skills which may influence results; also peer pressure is still an issue.

---

<sup>27</sup> This age (7) is based on research from the United States of America and Western Europe which have privileged circumstances and in less privileged countries/areas the age of this major developmental milestone may be higher. Also not all children develop at the same pace so this could also affect this age.

<sup>28</sup> They consider children below this age as incapable of being questioned as they do not have these skills. However NCB (2011, p.17) say even children under 5 can be involved in basic research if the methodology is adjusted for their age group to take into account things like their limited attention spans.

<sup>29</sup> Some students may develop slower than others so ages are just a guide and research must consider less developed students.

<sup>30</sup> For example they probably won't understand negations (such as not) or logical operators (like 'and', 'or' etc.)

<sup>31</sup> The age group the students of the pilot case study are in.

<sup>32</sup> They may provide answers that are popular with the group rather than their own opinion out of a desire to be popular or fear that their opinion will be unpopular. Providing privacy while conducting surveys, interviews etc. can help students feel able to resist peer pressure and provide their own opinion.

The National Children's Bureau "NCB" (2011, p.17) agrees with these views and believe most research methods can be used with secondary school students if they are adapted to meet their abilities (literacy levels, cognitive skills and understanding of abstract concepts).

Punch (2002) discusses how research with children can be different to research with adults and many approaches can be taken because of this. Children may be viewed the same as adults or completely differently; if it is the former the same methods are used with adults and children and no special treatment or consideration should occur with children<sup>33</sup> whereas with the latter adjustments should be made. Such adjustments are made based on observations of the children and the skills of their age group. However adults may struggle to create appropriate research involving children as adults find it difficult to think like a child. Punch also suggests a "best of both" approach that considers children as being similar to adults but uses research methods specifically designed around children and their skills<sup>34</sup>. They discuss similar areas to aforementioned research<sup>35</sup> around how children are different from adults and issues conducting research which need to be addressed. This includes: it is important to be impartial, children are easily influenced, answers may not be valid or reliable as children are more likely to lie (due to peer pressure, to avoid difficult subjects, to provide a perceived popular or correct answer and so forth), use appropriate research methods, and language needs to be adjusted to be clear/understandable.

## **5.6. Research methods and methodologies**

As this research is based around secondary school students, with the pilot case study being trialled with year 8 (12 to 13 year olds), most research methods can be used; however consideration will need to be made to ensure it is understandable to the students and validity of responses is considered.

To provide data to assess whether the aims and objectives have been met will require a mixed method approach of both qualitative and quantitative research. The majority of data around students' perceptions and understanding of computing will be quantitative and could be gathered via survey research with questionnaires or interviews. Other data such as student and teachers opinions will be qualitative and can be gathered via smaller interviews, observations or perhaps non-numeric attitudinal responses in a survey. Some data collected will be more in-depth, such as enquiring about computing influences, reasons for interests in computing

---

<sup>33</sup> Children should be considered as mature, capable and knowledgeable so require no special treatment.

<sup>34</sup> This is similar to the aforementioned research by De Leeuw (2011) and the National Children's Bureau (2011)

<sup>35</sup> De Leeuw (2011) and the National Children's Bureau (2011)

careers and observations of students while they work through the events tasks, so require qualitative methods. The triangulation of both qualitative and quantitative research methods will allow for the weaknesses of each method to be cancelled out (Dawson 2007, p.22).

There are a number of approaches to the research that could be considered. Dawson (2007) highlights action research, ethnography and grounded theory specifically. The aim of the research is to see if interactive teaching methods can improve students' perceptions and understanding of computing.

The research could be considered as action research as it involves close collaboration with schools to conduct research and to help improve students' perceptions and understanding of computing. However the researcher is not directly working with the students who are the main respondents and there is no close collaboration with the students. The research is not ethnography as it is not looking at a group's behaviour/cultural phenomena via observations over time.

Grounded theory involves researching an area and finding out about perceptions and thoughts, often from deep interviews and analysing the resultant data, before looking at the literature. The literature has already been examined in this work so grounded theory will not be appropriate.

There are many research methods that can be considered such as experiments, case studies, surveys and interviews. An experiment was considered but these require the researcher to have control over all the behaviours/variables (Yin 2003, p.8) so that the only behaviour/variable that can change is the one that is being tested. Therefore as this research has many possible areas being investigated this is not suitable especially as there are areas which are not easily defined such as the definition of fun which requires multiple aspects to be investigated to try and interpret the effectiveness of the event.

Case studies do not require control over all the behaviours/variables as they are designed to allow observations of multiple results so are more suitable for this research. Case studies are ideal for this research as they investigate contemporary phenomenon in a real-life context (Yin 2003, p.13) which in the case of this research is computing in schools. They can also handle multiple data points, from potentially multiple sources of evidence from different data collection methods, contributing to one result which this research will create, as for example

many questions will be required to test if the aims have been met. Also as the research is a defined repeatable event, the case study can be repeated many times with adjustments made if required. The repeated case studies can become part of a larger multiple-case study to offer more detailed results and room for analysis<sup>36</sup> with a larger or more varied results set as required<sup>37</sup>. Case studies also allow for various data collection methods to be used such as surveys, observations and interviews so will allow the research to collect the required qualitative and quantitative data.

Data can be collected in various ways such as surveys, observations, focus groups and interviews and the type chosen depends on the data to be collected and the audience. As explained in section [5.5](#) care must be taken when choosing the research methods used with children. Surveys are suitable for the age group involved as long as the questions are carefully written and are unambiguous. Surveys can allow for both quantitative and qualitative data to be easily collected from multiple people in a short time. Therefore it is ideal for this research which involves collecting a wide variety of data from a group of respondents. Also there is likely to be limited time to conduct the data collection and it must be simple for students to understand.

Observations are suitable if they are unobtrusive and do not cause any distress. Observations are ideal when participants are involved in activities as they allow data to be collected about how effective the activities are and the participants' abilities in completing them.

Interviews may be feasible if they are carefully constructed and questions used are clear; however this is not feasible with all the students due to time restrictions but it could be considered as either a separate study or with just a few students if there is spare time and the school feels it is appropriate.

---

<sup>36</sup> It can be argued that it is difficult to generalise results from single case studies, the same problem also occurs with single experiments, (Yin 2003, p.10) and using multiple case studies solves this problem.

<sup>37</sup> For example the same case study repeated without any changes can be used for validation to see whether noticed observations/results repeat and thus are reliable or were purely coincidental. Alternatively changing variables between case studies allows for testing to see if it changes results; it is recommended to do multiple cases of each variable change and the original unchanged version to validate results.

A focus/discussion group could be used to obtain opinions from the group in a shorter time than multiple individual interviews but this is unlikely to produce as reliable or in-depth responses. This is because children are easily influenced by peers and wish to give popular answers so will only say what they think others want to hear<sup>38</sup>.

Although interviews are not likely to be feasible with students they can be used with school staff to get their opinions on the event and students' abilities, or alternatively this can be achieved via a discussion.

## **5.7. This research**

### **5.7.1. Participants**

Due to the time constraints of limited time to conduct the pilot event/case study and school summer holidays it was decided for the pilot event that year 8 students from St Edward's school and Poole High school would be invited. It was possible to run a half day version of the event at St Edward's school for year 8 students. Therefore this opportunity will be used to create a pilot event with a selection of tasks and challenges to fit into half a day.

### **5.7.2. Teams**

The students will be put into teams of varied skills and include students from different schools<sup>39</sup>. This will help with inclusiveness and differentiation as teams will have a variety of skills and therefore everyone can learn from each other. Mixing teams up between schools and skill levels will also provide more equal skills amongst teams and no teams should have an unfair advantage. The use of teams also reduces the amount of resources required. One thing to consider with teams is how to ensure all team members are involved as some of the team may be more confident and do the majority of the work; consequently some students may feel they are not as skilled as others in the team and feel they should let them take over the work or perhaps they are unmotivated or lazy and do not wish to work.

Teams will probably work best for the more advanced tasks and the challenges where the students can work together to work out what needs to be done and different ideas are used to solve problems. However basic tasks should probably be done by individuals or pairs as there may not be enough work for a team to divide between themselves<sup>40</sup>.

---

<sup>38</sup> One of the general disadvantages of focus groups is participants may feel uncomfortable speaking in a group (Dawson 2007, p.31); this is especially a problem when working with children.

<sup>39</sup> However the event could be run with just 1 school as was the case with the pilot event/case study.

<sup>40</sup> Resource limitations may however make this unfeasible.

### 5.7.3. Content

Microcontrollers and microcomputers were considered to integrate physical computing into some of the tasks, see section 4. Raspberry Pi computers were chosen due to their: popularity in schools<sup>41</sup>, low cost, specifications<sup>42</sup>, and because they have General Purpose Input/Output (GPIO) pins which can be used by a variety of programming languages to control electronic components.

Programming languages/tools/environments were considered with their suitability for interacting with external devices and GPIO pins via a Raspberry Pi computer, see section 3. Scratch was chosen because: the students attending the event will already have some experience in using it, it is simple and doesn't use complex syntax, it is motivational and has graphical outputs, it can be used to quickly and easily demonstrate programming concepts, and it can be used to control GPIO pins on a Raspberry Pi.

### 5.7.4. Tasks

**Part 1:** Basic programming tasks with Scratch covering fundamental programming concepts with examples that can be expanded on in part 2.

**Part 2:** Further Scratch tasks which use a modified version of Scratch to interact with the GPIO pins of a Raspberry Pi for creating inputs and outputs of the program.

Students work through as many tasks and challenges within each part as they can which increase in complexity and make use of skills and concepts learned. This will allow students/teams to progress at their own pace and those with more advanced skills will have extra more challenging tasks to progress on to and keep them motivated. The more advanced tasks have less detail to be a problem solving challenge. More details are in appendix 7.

### 5.7.5. Further details

For further details on the event see:

- Appendix 8 - The event's timetable
- Appendix 9 - Tasks Worksheets/Hand-outs
- Appendix 10 – Challenges Worksheets/Hand-outs
- Appendix 11 – Advice for event staff
- Appendix 12 – Guidance for the event organiser

---

<sup>41</sup> The Raspberry Pi was primarily designed for education and many schools including St Edward's are investing in some for their classrooms. Therefore their Raspberry Pi computers can be used for the pilot event/case study.

<sup>42</sup> It has the fastest CPU, most memory, best graphics and so forth of the microcontrollers and microcomputers considered and many other useful features such as a wide variety of connections.



## **5.8. Data collection**

### **5.8.1. Surveys**

Surveys will be conducted before and after the event to see if the event enhanced students' perceptions and understanding of computing and met the aims and objectives. See appendix 13 for more details.

Prior to the event the students will be asked to complete a survey asking about their current perceptions and understanding of computing and their computing influences. After the event, in addition to repeating the previous survey questions which relate to students' perceptions and understanding of computing to see if they were improved, additional questions will be asked on their impression of the event. Teachers will also be asked after the event for their views on the events effectiveness.

Most questions will have 1 - 5 point answers in a likert scale. For these questions radio buttons for each value will be used if completed online or tick/check boxes if paper-based. Some questions will use text boxes for more flexible answers. Some other questions will have options to choose from as their answers and use radio buttons or tick/check boxes<sup>43</sup> depending on if multiple options are applicable. The majority of the questions will be closed-ended as they offer set answers to choose from but some such as "write 3 words that describe your opinion of computing" will be open-ended allowing any words to be written.

### **5.8.2. Observations**

In addition to the formal surveys the observers (those helping run the event) will be asked to provide feedback on the effectiveness of the day. See appendix 14 for more details.

## **5.9. Ethics for surveys, observations, interviews and discussions**

To meet the Bournemouth University (BU) Research Ethics Code of Practice (BU 2009) and taking advice from the Guidelines for Research with Children and Young People (NCB 2011) the following practices will be followed.

All content used with students will be in plain/simple language which they will be able to understand.

Results will be presented in an accessible format for young people to understand should they wish to read the findings.

---

<sup>43</sup> Boxes to tick will be used if paper-based survey is used.

The purpose of the research will be explained to students and school staff involved prior to research beginning and the author's email address will be provided should they have any further questions. It will be made clear why their input is valuable and that they are not obligated to take part in any of the research and can opt-out of any part of it. All data collected from the students will be anonymous and kept secure.

The research will be agreed with the relevant staff at the schools "the gatekeepers" who will be asked to ensure their students are comfortable with taking part in the research and that they understand they can opt-out of any part of it<sup>44</sup>. A research information sheet (appendix 15) will be provided to assist with this process. Staff will also be informed of the importance that they remain neutral to avoid affecting the results. The gatekeepers' permission will be the primary permission for the research to proceed with the students able to opt-out if they wish. Participation in the surveys will assume consent to publish data anonymously and this will be explained at the start of each survey. Gatekeepers will also be present throughout the research to ensure the students are happy to continue with it.

Gatekeepers will also be asked to seek permission from parents/guardians if they feel it is necessary.

Permission will also be gathered to publish any data/information from discussions with school staff and this will not contain any personal data or data which can identify students.

As the research is based around tasks which are similar to lesson content that students are familiar with it should not distress them. Also the student surveys will be simple and easy to understand/answer and their responses will be anonymous. Additionally, observations of the students during the event will be unobtrusive and their purpose will be explained which should also remove any potential distress.

---

<sup>44</sup> The importance of voluntary consent by the students will also be explained to the gatekeepers for when they explain the research to them.

## 6. Case study

### 6.1. Introduction

The pilot event/case study was a half day version of the outreach event, hosted at St Edward's School, with a group of 14 year 8 students. Due to the students' previous experience using Scratch, their teacher (Alastair Barker) thought they would not need the introductory tasks so part 1 was removed and thus the event fitted into half a day<sup>45</sup>. Alastair and a few of his colleagues attended the event and assisted the students.

### 6.2. Analysis of the event

#### 6.2.1. Content

Alastair explained how he thought there was too much text on the worksheets which most students wouldn't read and it is probably too complex for many of the students. He recommended going through the tasks with the students instead. As there was limited time it was decided to take this advice and show the students how to do complex parts of the tasks as well as providing worksheets for guidance and to allow the students to do as much as they can themselves. This proved to be a useful strategy as it became clear that the researcher had overestimated the students' abilities and they required more assistance than anticipated. The text on the worksheets was indeed too detailed for some students so should be simplified if the event is repeated.

The introductions presentation had too much content which was of little interest to the students but is useful for giving a background to the research and the tasks. Therefore the presenter should assess which content is applicable for the group such as are they interested in learning more about the research and content associated with the tasks, such as a background to electronics, and if not then reduce this content.

---

<sup>45</sup> The survey was adjusted accordingly such as removing the question asking about opinions of part 1.

The first 2 tasks (see appendix 9) went well although many students required assistance and tasks were explained by the researcher rather than the students only following the worksheet. Some had more confidence and abilities and were able to follow the worksheets and they quickly completed the tasks which provided them time to experiment with the code and electronics in order to discover what else they could do with it. When the electronics got more complex in task 3 with the use of multiple GPIO pins and wires it was too confusing for many students and their attention/motivation was lost. Alastair therefore recommended ending the event as the students had lost interest. This was unfortunate as it meant only 2 of the 3 tasks had been completed and none of the challenges were attempted. The difficulties encountered and reduced content covered meant many of the concepts that the researcher aimed to cover were not fully explained and there was less opportunities for learning. As the event didn't run smoothly and some students had difficulty understanding some of the content it probably reduced the chances of improving opinions of computing and increasing knowledge. It would be advisable, if repeated, to split the tasks up into smaller stages and simplify the electronics.

As there were enough Raspberry Pi computers for one per person this was used to ensure all students were involved. Teams are more useful for when students are working unassisted and for students to learn together so this was used for the more complex tasks<sup>46 47</sup>.

### 6.2.2. Students

There was a broad mix of students with a variety of different skills and interests and some with learning difficulties. Whereas some were able to work on their own or with others, most needed help and the researcher and the school staff had to help students with most of the tasks.

The students had chosen to attend a computing event<sup>48</sup> so had previous interest in computing rather than using a random mix of students, they were also all male; therefore results could be slightly biased towards computing being popular.

---

<sup>46</sup> This is also useful to overcome an oversight of there not being enough jumper wires/leads purchased for the tasks which required 6 jumper wires so students worked in teams of 2 or 3 for these; however most tasks only required 2 wires and there were enough wires for them to work individually on these tasks.

<sup>47</sup> Teams would have also been used for completing some of the challenges if it had been possible to include them

<sup>48</sup> They originally registered for a minecraft event which was unfeasible and this event was added as a backup

### 6.2.3. Surveys

The before and after event surveys were prepared to be completed online but as the computers the students would be working on (the Raspberry Pis) didn't have internet access it was decided it would be better to use a paper-based survey<sup>49</sup>. Unfortunately a paper-based survey hadn't been prepared<sup>50</sup> but the list of questions was available which was photocopied and the students were asked to write on the sheet. Unfortunately this meant it was possible to ignore questions whereas the online survey would have enforced completeness. For the after event survey the students used the same survey/form and were asked to add new scores for the relevant questions<sup>51</sup> to see if their opinions and understanding had improved. It had originally been planned to run the before event survey at least a week before the event so there was a gap between surveys but this wasn't feasible so it was conducted at the beginning of the event. The short time between surveys and the use of the same form may have affected results as there was little time for the students to consider the effect the event had on their views and made duplicating results from the before survey easy<sup>52</sup>.

Also due to the lack of interest by the end of the event it was decided not to add the additional after session questions about their views of the event.

### 6.2.4. Discussion

As some students struggled with the tasks and needed assistance they would probably benefit from simpler tasks. However some students had no problems and were easily able to work independently through the worksheets showing that the content is suitable for them. Therefore it is advisable to make the content flexible with more basic content to use as an introduction and progressing onto more advanced content thus allowing students to progress at their own pace. Coincidentally this was planned for the full event but the researcher was advised that the introductory content was not required for the half day pilot event. Perhaps more flexibility should be included to allow for tasks to easily be chosen from a range of tasks during the event in order to cater for the needs and skills of the students.

---

<sup>49</sup> There were other computers in the room with internet access but Alastair recommended not using them as it would be too time consuming and disjointed to require students to switch computers to do the surveys.

<sup>50</sup> For future sessions this should be created as a backup solution.

<sup>51</sup> Some questions only needed asking once (for example asking about family members who work with computers) as their answers will not have changed as a result of the event.

<sup>52</sup> For future events there will be gaps between surveys and the use of different forms (ideally online to enforce mandatory questions).

One sign that the event increased interest in computing including physical computing was that many students were interested in buying Raspberry Pi computers for further experimentation and learning.

Due to the complexity and difficulties found with using electronic components with the Raspberry Pi its suitability could be questioned; this is discussed in more detail in appendix 16.

## 7. Findings

The survey results are summarised below; see appendix 17 for full details.

### 7.1. Studying computing

The students were asked in a 1 to 5 scale (where 1 = very unlikely and 5 = definitely) questions about the likelihood of them studying computing.

First the students were asked how likely they would choose ICT or Computing as a GCSE option or an equivalent<sup>53</sup>; results from the before and after event surveys are shown in Figure 6 and the differences between surveys are highlighted in Figure 7.

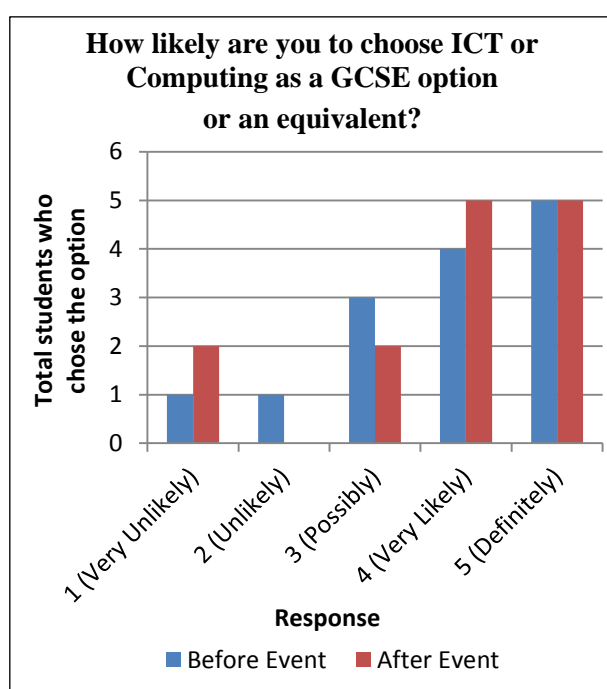


Figure 6: How likely students will choose ICT or Computing as a GCSE option or an equivalent

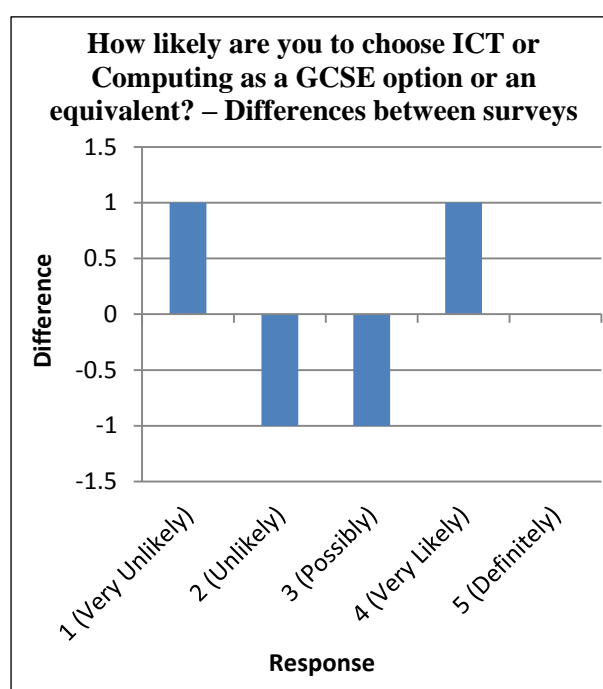


Figure 7: How likely students will choose ICT or Computing as a GCSE option or an equivalent - Differences between surveys

<sup>53</sup> The reference to ICT could be removed in future due to its removal from the National Curriculum in 2014. It can however still exist in schools that don't have to use the NC or in colleges; however given the negative perceptions of ICT it is probably best to remove it from the question.

The results show the subject is popular with 64.29% positive responses, 21.43% neutral responses and 14.29% negative responses<sup>54</sup>. After the event the positive responses increase to 71.43%, neutral reduces to 14.29% and negative remains the same. However you could consider ‘possibly’ as a positive response<sup>55</sup> and thus can combine the positive and neutral totals which in this case makes 85.72% for both surveys despite slight decreases and increases in responses<sup>56</sup>.

The quartiles and averages (Table 3 and Table 4, and Figure 8) also show the increase in positivity after the event with a higher first quartile, however the mode average decreases by 1. The differences between surveys are not statistically significant at the 5% level<sup>57</sup> (U= 95.5, Z=-0.0919, P= 0.92828).

Table 3: Quartiles for the “How likely students will choose ICT or Computing as a GCSE option or an equivalent” question

	<b>Before Event</b>	<b>After Event</b>
<b>Minimum</b>	1	1
<b>Quartile 1</b>	3	3.25
<b>Median (Q2)</b>	4	4
<b>Quartile 3</b>	5	5
<b>Maximum</b>	5	5

Table 4: Averages for the “How likely students will choose ICT or Computing as a GCSE option or an equivalent” question

	<b>Before Event</b>	<b>After Event</b>	<b>Difference</b>
<b>Mean</b>	3.79	3.79	0.00
<b>Median</b>	4	4	0.00
<b>Mode</b>	5	4	-1

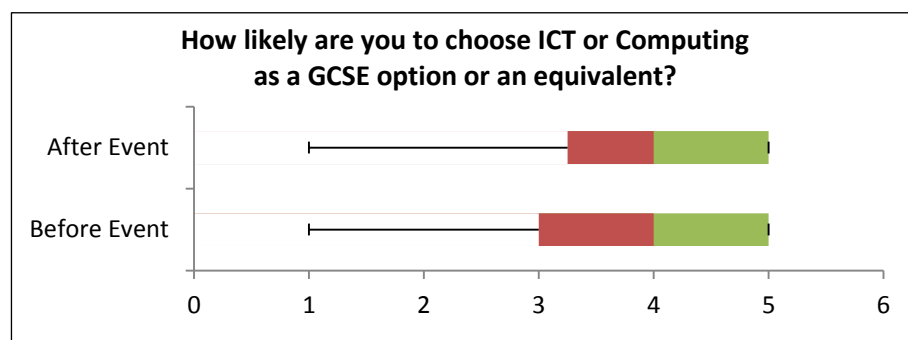


Figure 8: Box plot for the “How likely students will choose ICT or Computing as a GCSE option or an equivalent” question

<sup>54</sup> Positive responses are ‘very likely’ and ‘definitely’. Possibly is neutral as it is neither positive nor negative and ‘unlikely’ and ‘very unlikely’ are negative responses.

<sup>55</sup> It could be considered positive as they may choose the option.

<sup>56</sup> There is an additional ‘very unlikely’ response and one less ‘unlikely’ but also one less ‘possibly’ and one more ‘very likely’.

<sup>57</sup> Statistical significance throughout this dissertation uses the Mann-Whitney-Wilcoxon/Mann-Whitney U test at the 5% level. They are two-tailed tests unless otherwise stated. There were 14 responses in each sample (before and after event surveys) unless otherwise stated; some questions were ignored by some students so have less responses which are explained with their results including the reduced totals.



When we look at the likelihood of students studying AS/A level Computing or a college computing course<sup>58</sup> (see Figure 9 and Figure 10) responses are varied with mostly positive options chosen (42.86% positive, 28.57% neutral, 28.57% negative) with a clear improvement as a result of the event (57.14% positive, 14.29% neutral, 28.57% negative)<sup>59</sup>. However the averages are lower than for the GCSE question. This reduced certainty could be because AS/A levels are further into the future and students haven't thought about such decisions yet<sup>60</sup>.

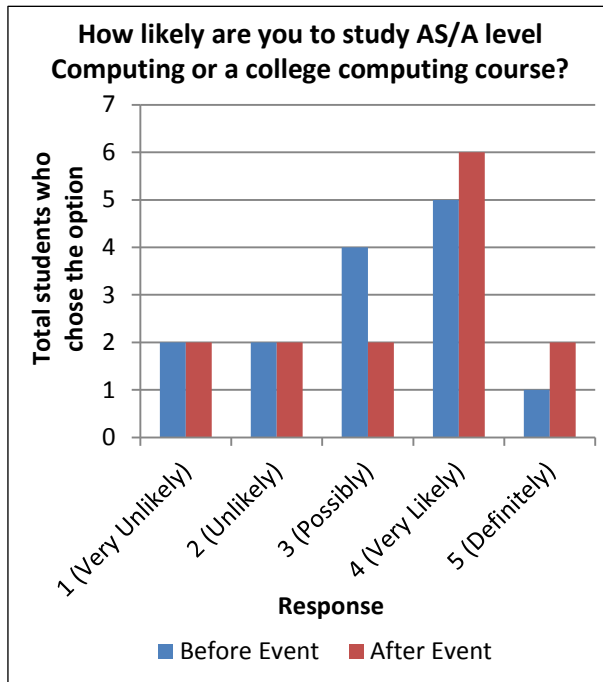


Figure 9: How likely students will study AS/A level Computing or a college computing course

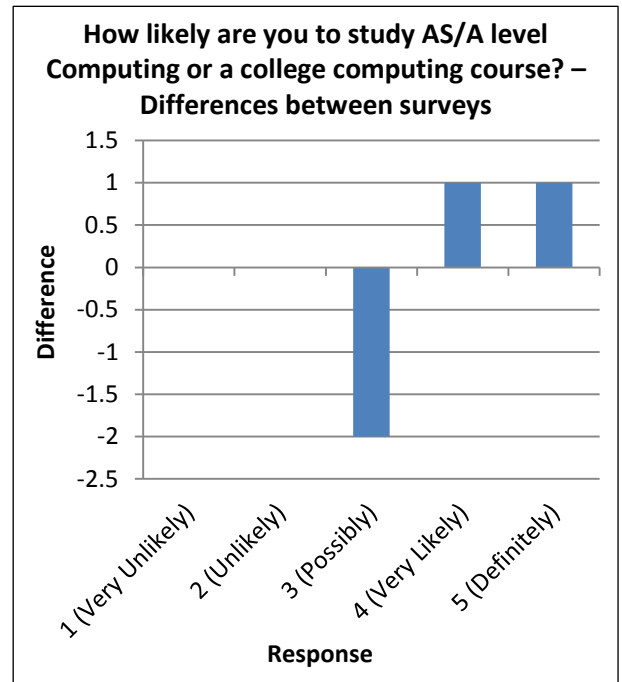


Figure 10: How likely students will study AS/A level Computing or a college computing course - Differences between surveys

<sup>58</sup> “How likely are you to study AS/A level Computing or a college computing course?”

<sup>59</sup> There are increases for the more positive options (‘very likely’ and ‘definitely’) with less choosing the ‘possibly’ option showing increased opinions. However the ‘very unlikely’ and ‘unlikely’ options remain unchanged which is encouraging for showing no decline but equally these figures didn’t decrease.

<sup>60</sup> The students in the pilot case study were in year 8 (12-13 years old) who won’t have to decide on AS/A Level options until year 11 (15-16 year olds) so will have many more years to decide on subject choices.

The quartiles and averages (Table 5 and Table 6, and Figure 11) shows the results between surveys are similar but there are increases for the mean and median averages. The differences between surveys are not statistically significant ( $U= 86, Z= -0.5284, P= 0.59612$ ).

Table 5: Quartiles for the “How likely are you to study AS/A level Computing or a college computing course” question

	Before Event	After Event
<b>Minimum</b>	1	1
<b>Quartile 1</b>	2.25	2.25
<b>Median (Q2)</b>	3	4
<b>Quartile 3</b>	4	4
<b>Maximum</b>	5	5

Table 6: Averages for the “How likely are you to study AS/A level Computing or a college computing course” question

	Before Event	After Event	Difference
<b>Mean</b>	3.07	3.29	0.21
<b>Median</b>	3	4	1.00
<b>Mode</b>	4	4	0

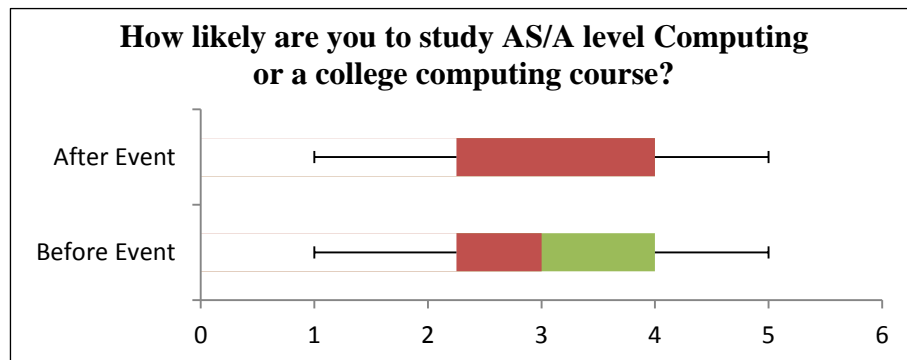


Figure 11: Box plot for the “How likely are you to study AS/A level Computing or a college computing course” question

Looking further into the students’ future they were asked about their likelihood of studying a computing course at university. Results (Figure 12 and Figure 13) show a reasonable amount of interest in computing at university with the majority of students providing a positive response (57.14% positive, 21.43% neutral and 21.43% negative)<sup>61</sup>. Popularity decreases slightly as a result of the event<sup>62</sup> making responses 50% positive, 28.57% neutral and 21.43% negative.

<sup>61</sup> Interestingly positive responses are higher than those considering a AS/A level Computing or a college computing course but slightly less than for the responses about considering a computing GCSE or similar. However this isn’t surprising as they will have to consider GCSE options far sooner than university subject choices.

<sup>62</sup> There was one less response for ‘very likely’ and one more for ‘possibly’ resulting in a slight decrease for positive results and a slight increase for neutral.

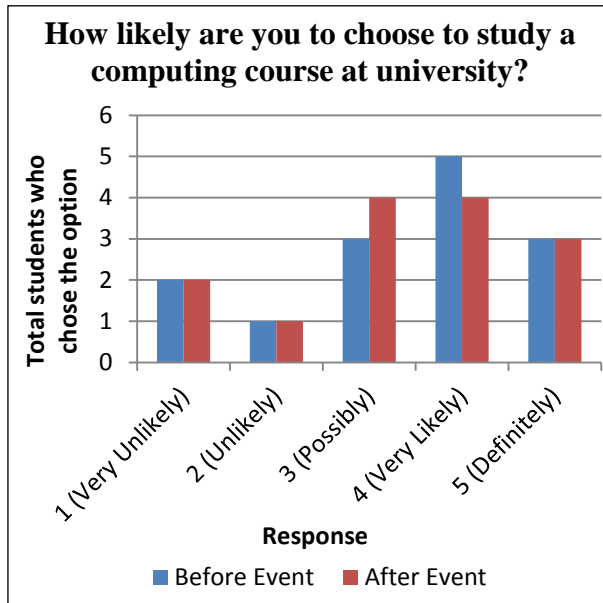


Figure 12: How likely students will choose to study a computing course at university

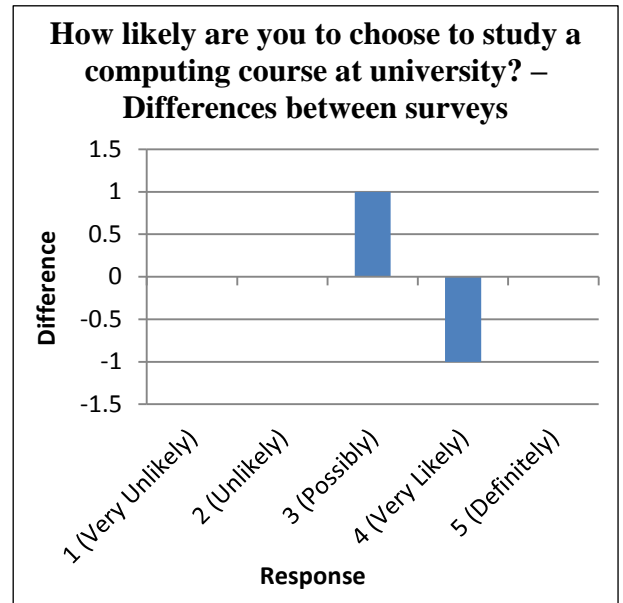


Figure 13: How likely students will choose to study a computing course at university – Differences between surveys

The averages and quartiles for the surveys (Table 7 and Table 8, and Figure 14) highlights the positive attitudes to university and the slight decrease in positivity after the event. The differences between surveys are not statistically significant ( $U= 94, Z= 0.1608, P= 0.87288$ ).

Table 7: Quartiles for the “How likely students will choose to study a computing course at university” question

	Before Event	After Event
<b>Minimum</b>	1	1
<b>Quartile 1</b>	3	3
<b>Median (Q2)</b>	4	3.5
<b>Quartile 3</b>	4	4
<b>Maximum</b>	5	5

Table 8: Averages for the “How likely students will choose to study a computing course at university” question

	Before Event	After Event	Difference
<b>Mean</b>	3.43	3.36	-0.07
<b>Median</b>	4	3.5	-0.50
<b>Mode</b>	4	3	-1

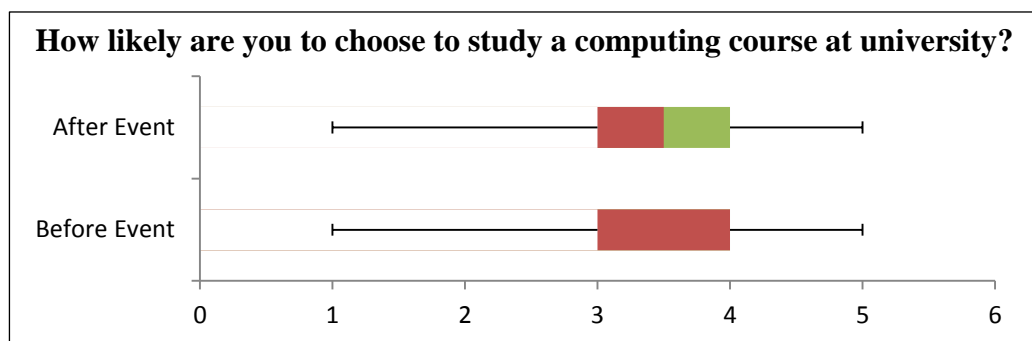


Figure 14: Box plot for the “How likely students will choose to study a computing course at university” question

## 7.2. Career ambitions

Various questions were asked about students career ambitions. First they were asked how likely they would get a job in the computing industry and the results are shown in Figure 15 and Figure 16.

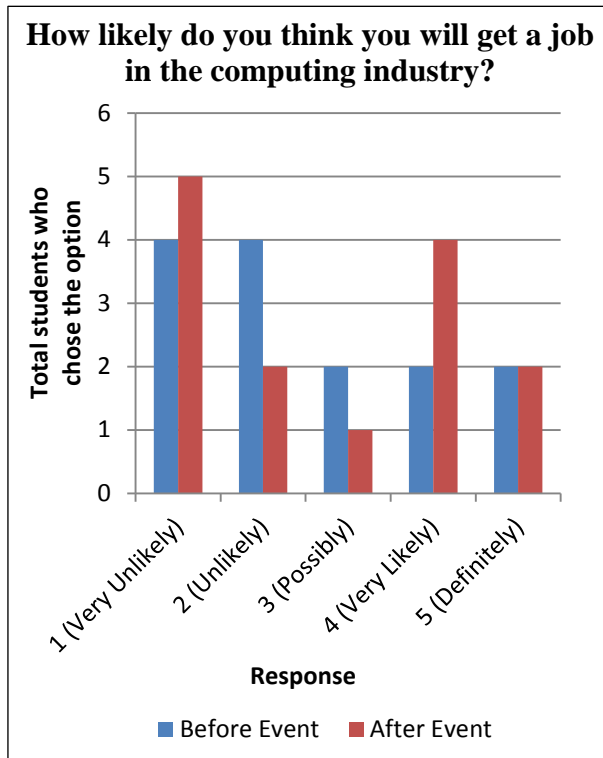


Figure 15: How likely students think they will get a job in the computing industry

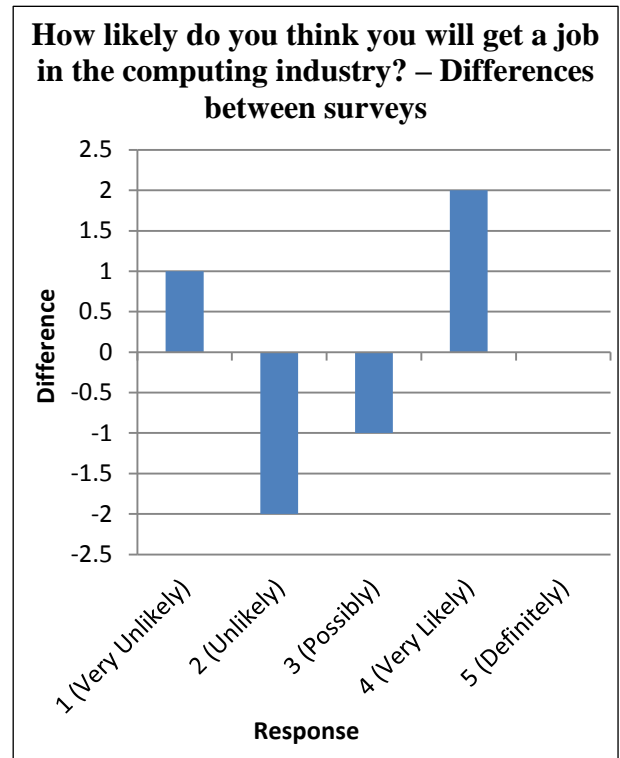


Figure 16: How likely students think they will get a job in the computing industry – Differences between surveys

This shows that before the event opinions were slightly negative overall (28.57% positive, 14.29% neutral, and **57.14% negative**) and opinions improved after the event with positive responses increasing to 42.86% and neutral and negative results decreasing to 7.14% and 50% respectively<sup>63</sup>.

<sup>63</sup> An encouraging sign was that the ‘very likely’ responses doubled and the ‘definitely’ responses remained the same (thus the increase in ‘very likely’ wasn’t at the expense of lost ‘definitely’ responses). However there was 1 extra ‘very unlikely’ response indicating a student is less certain working in the computing industry is right for them which is disappointing considering one of the event’s objectives was to increase interest in computing careers.

The averages and quartiles for the surveys (Table 9 and Table 10, and Figure 17) highlights the increased positivity (although the mode average decreased by 1 and first quartile decreased by 0.25) along with a wider range of quartiles. The differences between surveys are not statistically significant ( $U= 95, Z=-0.1149, P= 0.9124$ ).

Table 9: Quartiles for the “How likely do you think you will get a job in the computing industry” question

	Before Event	After Event
<b>Minimum</b>	1	1
<b>Quartile 1</b>	1.25	1
<b>Median (Q2)</b>	2	2.5
<b>Quartile 3</b>	3.75	4
<b>Maximum</b>	5	5

Table 10: Averages for the “How likely do you think you will get a job in the computing industry” question

	Before Event	After Event	Difference
<b>Mean</b>	2.57	2.71	0.14
<b>Median</b>	2	2.5	0.50
<b>Mode</b>	2	1	-1

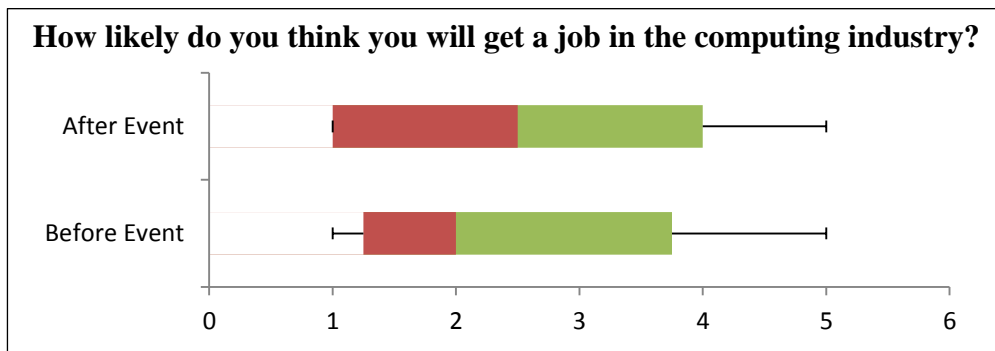


Figure 17: Box plot for the “How likely do you think you will get a job in the computing industry” question

Students were also asked whether they have considered working in the computing industry<sup>64</sup>. Unfortunately some students ignored the question<sup>65</sup>. The results (Figure 18 and Figure 19) remained the same between both surveys<sup>66</sup>. Of the 71.43% of students who answered the question 70% have considered working in the computing industry.

<sup>64</sup> This is similar to the previous question so could perhaps merge the questions for future uses of the survey, however it does work well to establish if the questions about their interest in the computing industry should be asked/answered (in the online version of the survey it only shows these additional questions if they answer yes).

<sup>65</sup> This was perhaps due to the use of the paper survey being confusing whereas if the online survey was used they would not be able to avoid questions.

<sup>66</sup> This is disappointing given one of the objectives of the event was to encourage more students to consider computing careers; however it didn't decrease its appeal either. It could be that the students were not interested in the after event survey and just repeated their answers from the before event survey.

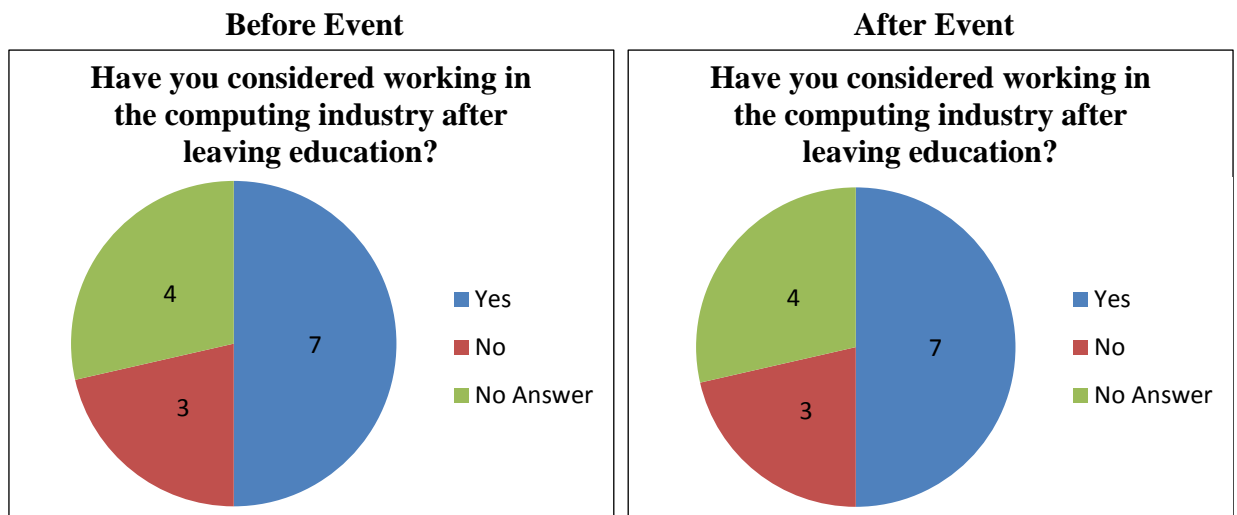


Figure 18: Students who have considered working in the computing industry after leaving education – Before event

Figure 19: Students who have considered working in the computing industry after leaving education – After event

The students who answered yes to this question were asked 2 additional questions about their interest in the computing industry. They were asked which sector they would like to work in (results are shown in Figure 20) and all respondents<sup>67</sup> chose Game Development and results didn't change between surveys. Perhaps this choice is indicative of the age group surveyed who frequently play games and may wish to be able to create them. Also they may not know much about the other sectors or they do not see them as “cool”<sup>68</sup>.

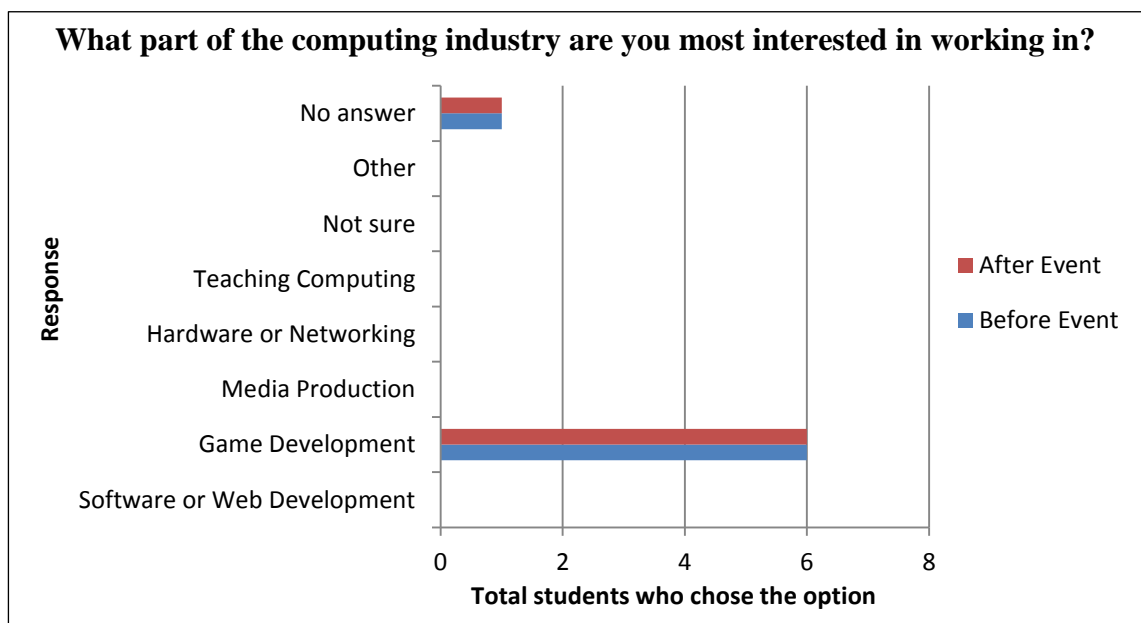


Figure 20: Sectors of the computing industry students are most interested in

<sup>67</sup> One student didn't answer this question.

<sup>68</sup> Children of this age group are easily influenced by what is popular at the time and may not be considering sectors in relation to jobs they could do and instead selected what they like now.

They were also asked about their motivation/reason for wanting to work in the computing industry (results are shown in Figure 21). The popular reasons chosen were working with computers would be cool and possible job prospects; also chosen was that the computing industry appears to be interesting and rewarding. Money was not a consideration. 2 students<sup>69</sup> didn't answer the question and also results were unchanged between surveys.

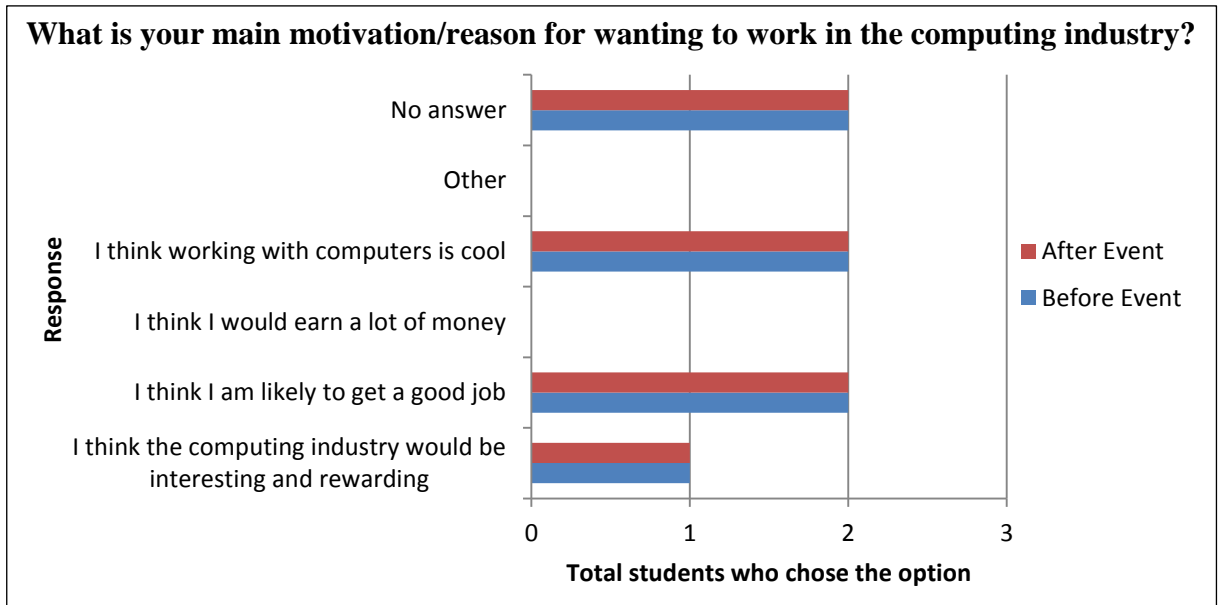


Figure 21: Students motivation/reasons for wanting to work in the computing industry

### 7.3. Computing skills

Some questions were used to assess the students' confidence in using and explaining programming languages, concepts and tools (these use a 1 to 5 scale where 1 = very unconfident and 5 = very confident). First they were asked about their confidence of explaining an 'if' statement; results are shown in Figure 22 and Figure 23.

<sup>69</sup> This is 28.57% of the total students who were eligible for answering the question (those who answered yes to whether they have considered a career in computing).

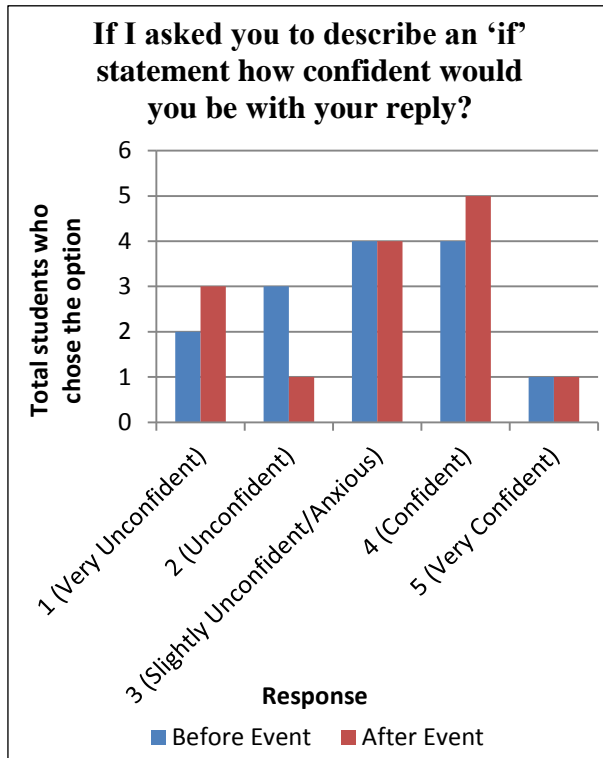


Figure 22: Students confidence around describing an 'if' statement

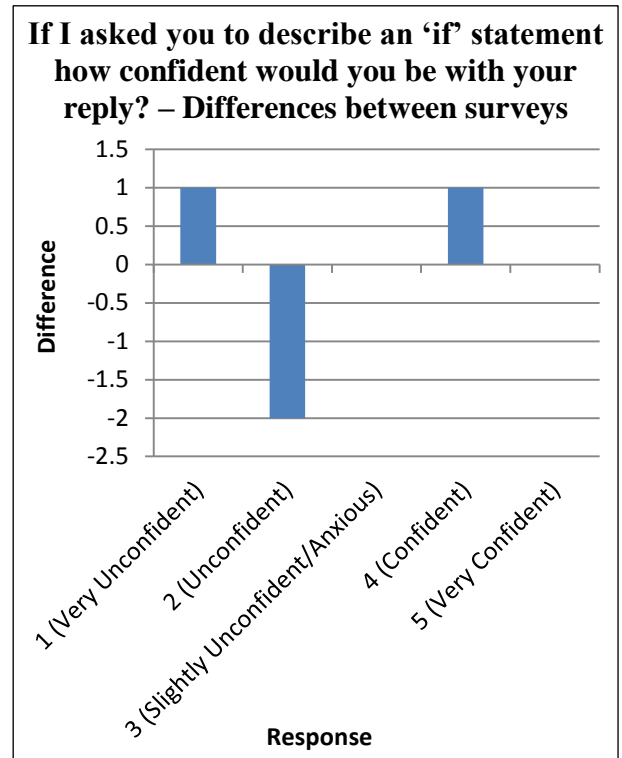


Figure 23: Students confidence around describing an 'if' statement – Differences between surveys

There is a reasonable amount of confidence explaining an 'if' statement with the majority of responses being positive (35.71%) or slightly negative (28.57%); negative responses was 35.71%<sup>70</sup>. After the event there was a slight increase in positive responses and a decrease in negative responses (42.86% positive, 28.57% slightly negative and 28.57% negative)<sup>71</sup>.

The averages and quartiles for the surveys (Table 11 and Table 12, and Figure 24) highlights the slight overall improvement in confidence as a result of the event (the mode and mean averages increased slightly). The differences between surveys are not statistically significant (U= 93, Z= -0.2068, P= 0.83366).

<sup>70</sup> 'Unconfident' and 'very unconfident' are classed as negative responses, and 'confident' and 'very confident' are positive responses. 'Slightly unconfident/anxious' is considered slightly negative as it isn't a positive response but isn't a completely negative response either. However as it is in the middle of the scale it could be considered as neutral like we did with possibly on the previous scale.

<sup>71</sup> There was one additional 'confident' response and 2 less 'unconfident' responses, however there was one additional 'very unconfident' response. Perhaps this was one of the reductions from the 'unconfident' responses, i.e. a student who previously said 'unconfident', now feels very unconfident describing 'if' statements, maybe the event confused them.



Table 11: Quartiles for the “If I asked you to describe an ‘if’ statement how confident would you be with your reply” question

	Before Event	After Event
<b>Minimum</b>	1	1
<b>Quartile 1</b>	2	2.25
<b>Median (Q2)</b>	3	3
<b>Quartile 3</b>	4	4
<b>Maximum</b>	5	5

Table 12: Averages for the “If I asked you to describe an ‘if’ statement how confident would you be with your reply” question

	Before Event	After Event	Difference
<b>Mean</b>	2.93	3.00	0.07
<b>Median</b>	3	3	0.00
<b>Mode</b>	3	4	1

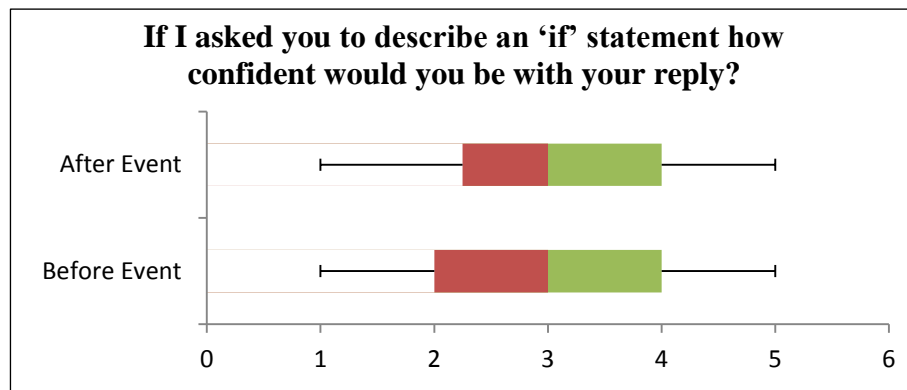


Figure 24: Box plot for the “If I asked you to describe an ‘if’ statement how confident would you be with your reply” question

The students were also asked about their confidence in explaining a loop; results are shown in Figure 25 and Figure 26. There was a reasonable amount of confident responses prior to the event but overall confidence was low (35.71% positive responses, 21.43% slightly negative, 42.86% negative) with various changes in responses in the after event survey<sup>72</sup> with less negative responses and more slightly negative responses<sup>73</sup> (35.71% positive, 28.57% slightly negative, 35.71% negative)<sup>74</sup>; overall confidence reduced.

<sup>72</sup> There was an increase in ‘very unconfident’ responses but less ‘unconfident’ responses and one more ‘slightly unconfident/anxious’ response.

<sup>73</sup> Although this appears to be a little more positive there was a large increase in ‘very unconfident’ responses which is a very negative change.

<sup>74</sup> Interestingly the positive and negative responses are now equal.

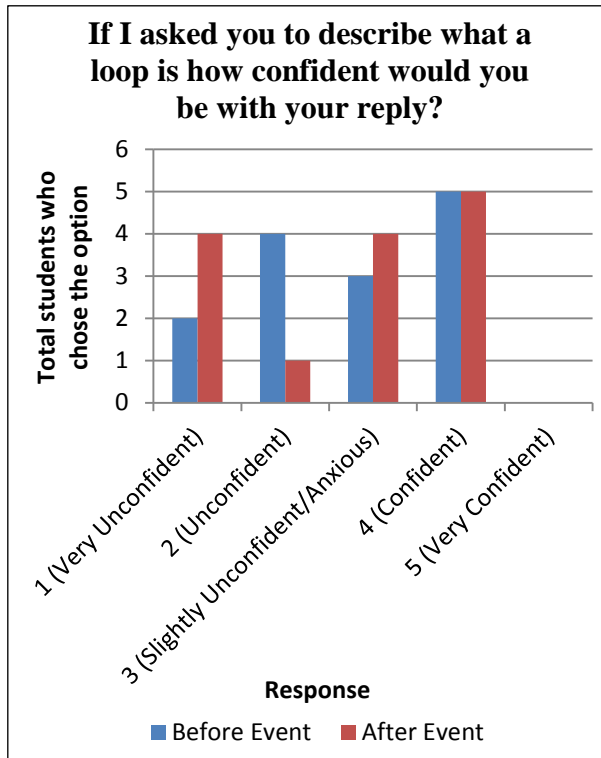


Figure 25: Students confidence around describing a loop

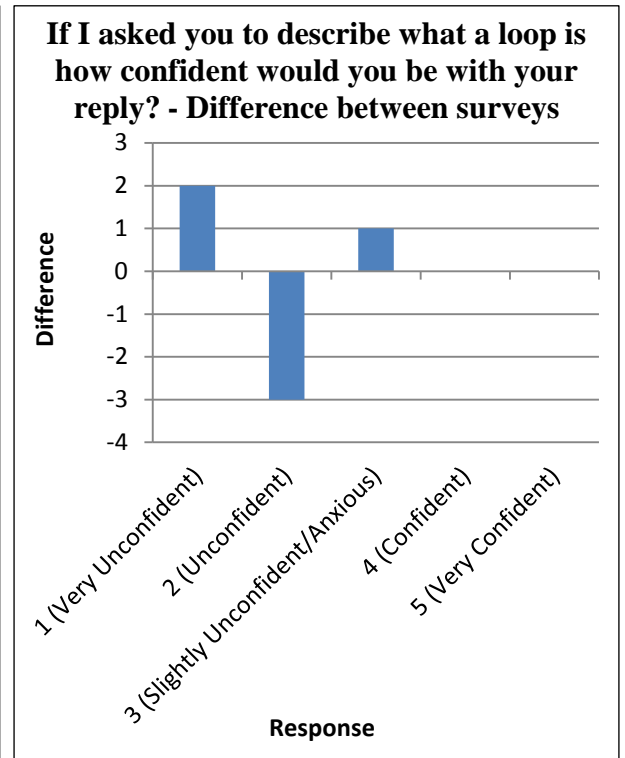


Figure 26: Students confidence around describing a loop – Differences between surveys

The averages and quartiles for the surveys (Table 13 and Table 14, and Figure 27) are very similar despite the differences in responses. However the after event survey has a lower first quartile and slightly lower mean average. The differences between surveys are not statistically significant ( $U= 95.5$ ,  $Z= 0.0919$ ,  $P= 0.92828$ ).

Table 13: Quartiles for the “If I asked you to describe what a loop is how confident would you be with your reply” question

	Before Event	After Event
<b>Minimum</b>	1	1
<b>Quartile 1</b>	2	1.25
<b>Median (Q2)</b>	3	3
<b>Quartile 3</b>	4	4
<b>Maximum</b>	4	4

Table 14: Averages for the “If I asked you to describe what a loop is how confident would you be with your reply” question

	Before Event	After Event	Difference
<b>Mean</b>	2.79	2.71	-0.07
<b>Median</b>	3	3	0.00
<b>Mode</b>	4	4	0

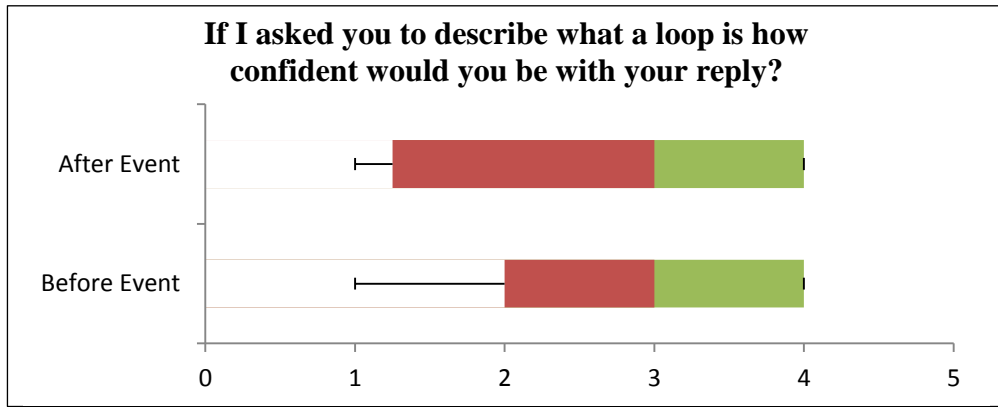


Figure 27: Box plot for the “If I asked you to describe what a loop is how confident would you be with your reply” question

The final programming concepts question looks at students’ confidence in explaining variables, results are shown in Figure 28 and Figure 29.

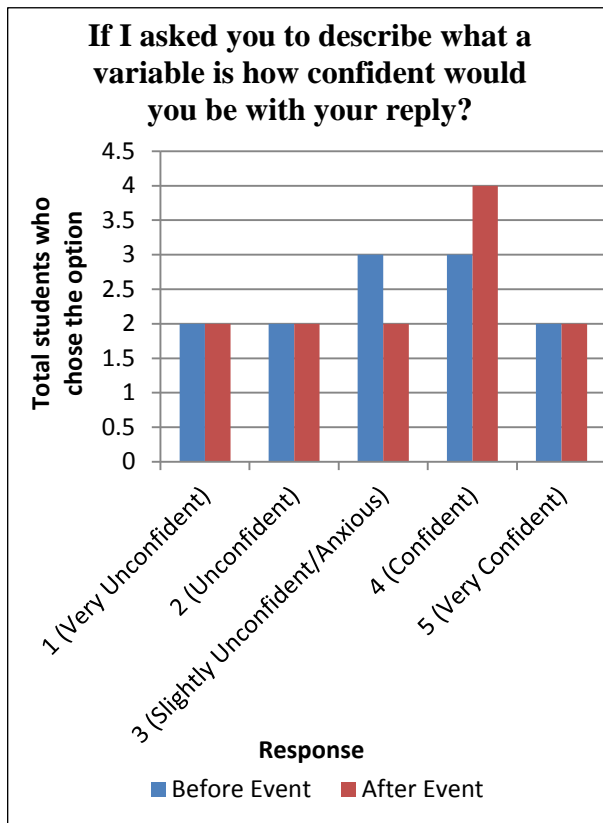


Figure 28: Students confidence around describing variables

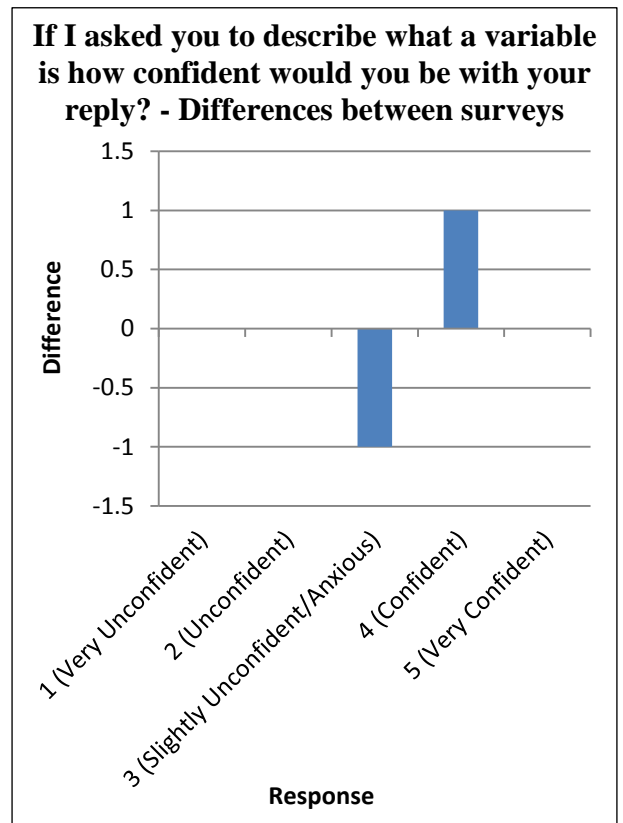


Figure 29: Students confidence around describing variables – Differences between surveys

Results<sup>75</sup> are almost identical between surveys with a slight increase in confidence after the event with a 1 response change from ‘slightly unconfident/anxious’ to ‘confident’<sup>76</sup>. This boosts positive results to 50%<sup>77</sup> but this is still a low amount.

The averages and quartiles for the surveys (Table 15 and Table 16, and Figure 30) also show responses are similar with slight increases for the after event survey. The differences between surveys are not statistically significant (U= 69, Z= -0.1443, P= 0.88866).

Table 15: Quartiles for the “If I asked you to describe what a variable is how confident would you be with your reply” question

	Before Event	After Event
<b>Minimum</b>	1	1
<b>Quartile 1</b>	2	2
<b>Median (Q2)</b>	3	3.5
<b>Quartile 3</b>	4	4
<b>Maximum</b>	5	5

Table 16: Averages for the “If I asked you to describe what a variable is how confident would you be with your reply” question

	Before Event	After Event	Difference
<b>Mean</b>	3.08	3.17	0.08
<b>Median</b>	3	3.5	0.50
<b>Mode</b>	3	4	1

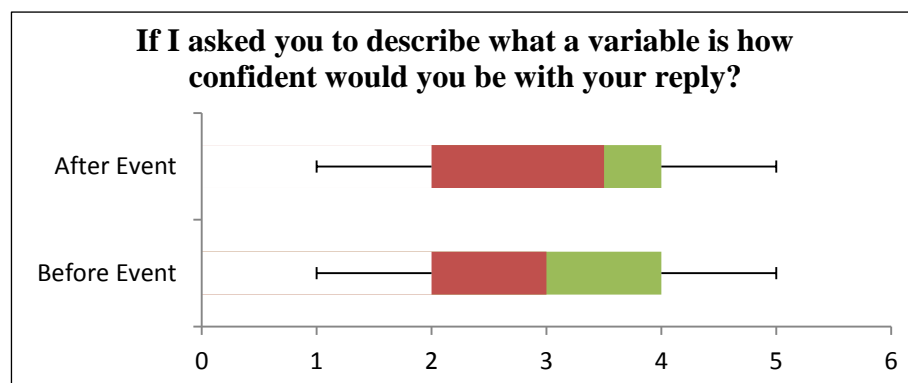


Figure 30: Box plot for “If I asked you to describe what a variable is how confident would you be with your reply” question

<sup>75</sup> 2 students didn’t answer this question thus there is only 12 responses instead of 14.

<sup>76</sup> It is highly likely this is just one person changing their response between these 2 options but it could be that multiple changes happened and the totals just happened to change overall in these 2 categories.

<sup>77</sup> Before the event the results were 41.67% positive, 25% slightly negative and 33.33% negative, and after the event it changed to 50% positive, 16.67% slightly negative and 33.33% negative.

Next they were asked about their confidence using Scratch to program; results are shown in Figure 31 and Figure 32.

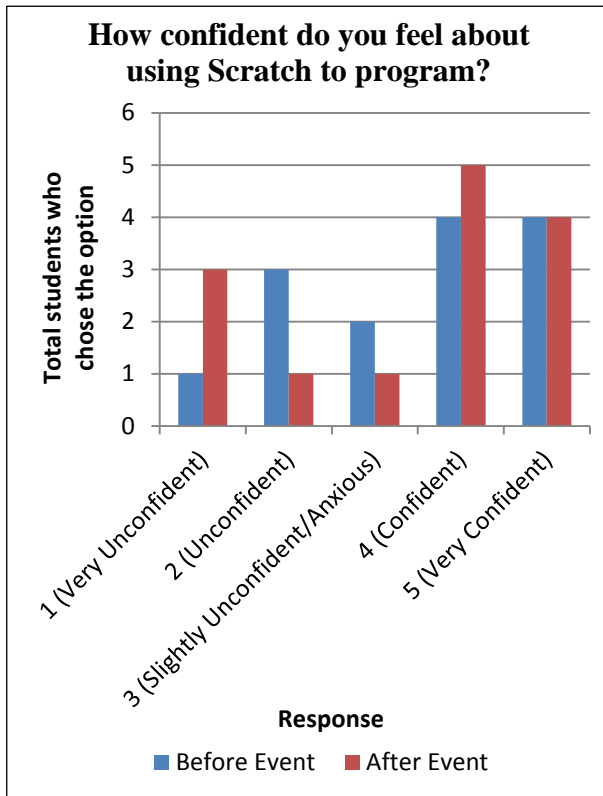


Figure 31: Students confidence around programming with Scratch

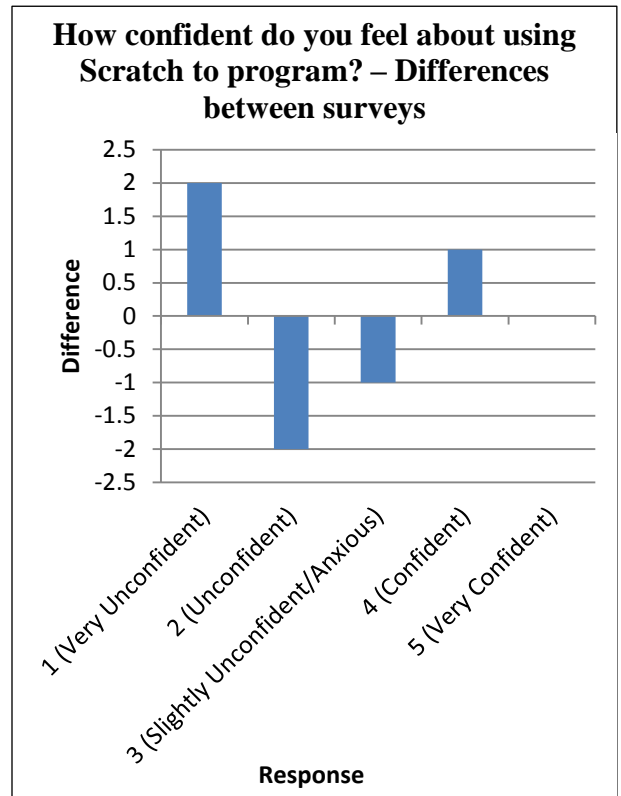


Figure 32: Students confidence around programming with Scratch – Differences between surveys

These graphs show that the event reduced some students' confidence to program using Scratch with a change of some responses to 'very unconfident'<sup>78</sup>; however there was an extra 'confident' response<sup>79</sup>. Overall confidence is reasonably high with positive responses of 57.14% before the event rising to 64.29% after the event<sup>80</sup>.

Despite the differences in responses the averages and quartiles for the surveys (Table 17 and Table 18, and Figure 33) are the same for both surveys with the exception of the mean average decreasing by 0.07 and the mode average decreasing by 1. The differences between surveys are not statistically significant (U= 97, Z= 0.023, P= 0.98404).

<sup>78</sup> These extra 'very unconfident' responses could have previously been 'unconfident' as these responses have reduced.

<sup>79</sup> There was also one less 'slightly unconfident/anxious' response so it could be that a student's confidence increased from this to confident.

<sup>80</sup> Responses were 57.14% positive, 14.29% slightly negative and 28.57% negative before the event and 64.29% positive, 7.14% slightly negative and 28.57% negative after the event.

Table 17: Quartiles for the “How confident do you feel about using Scratch to program” question

	Before Event	After Event
<b>Minimum</b>	1	1
<b>Quartile 1</b>	2.25	2.25
<b>Median (Q2)</b>	4	4
<b>Quartile 3</b>	4.75	4.75
<b>Maximum</b>	5	5

Table 18: Averages for the “How confident do you feel about using Scratch to program” question

	Before Event	After Event	Difference
<b>Mean</b>	3.50	3.43	-0.07
<b>Median</b>	4	4	0.00
<b>Mode</b>	5	4	-1

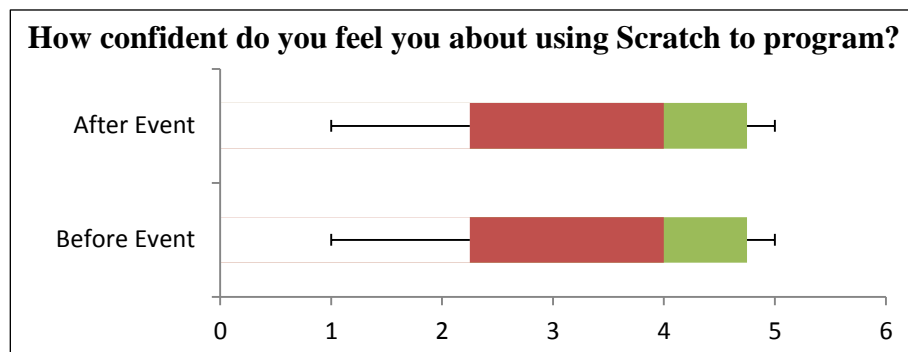


Figure 33: Box plot for the “How confident do you feel about using Scratch to program” question

They were also asked about their confidence learning new programming languages in order to establish if they feel confident enough with their programming skills and see similarities between languages that they feel able to learn new languages. The results are shown in Figure 34 and Figure 35.

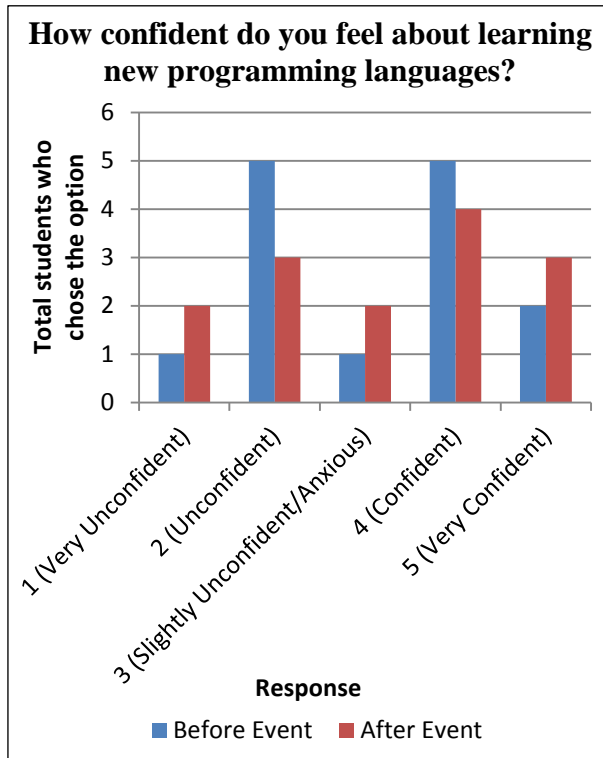


Figure 34: Students confidence learning new programming languages

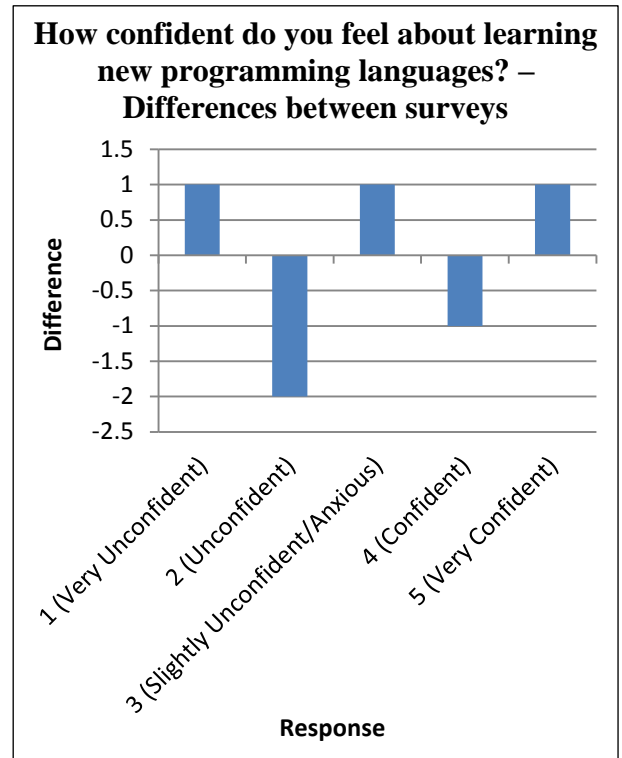


Figure 35: Students confidence learning new programming languages – Differences between surveys

These show a reasonably high confidence in learning new programming languages and there are many changes as an effect of the event with every category changing with overall less negative results chosen<sup>81</sup>. Responses were 50% positive, 7.14% slightly negative and 42.86% negative before the event and 50% positive, 14.29% slightly negative and 35.71% negative after the event.

Despite the differences in responses the averages and quartiles for the surveys (Table 19 and Table 20, and Figure 36) are the same with the exception of the mean average increasing by 0.07 and the mode average increasing by 2. The differences between surveys are not statistically significant (U= 94.5, Z= -0.1378, P= 0.88866).

<sup>81</sup> 50% of students said ‘confident’ or ‘very confident’ with an increase in ‘very confident’ responses after the event; there is also a reduction in ‘unconfident’ responses from 5 responses to 3.

Table 19: Quartiles for the “How confident do you feel about learning new programming languages” question

	Before Event	After Event
Minimum	1	1
Quartile 1	2	2
Median (Q2)	3.5	3.5
Quartile 3	4	4
Maximum	5	5

Table 20: Averages for the “How confident do you feel about learning new programming languages” question

	Before Event	After Event	Difference
Mean	3.14	3.21	0.07
Median	3.5	3.5	0.00
Mode	2	4	2

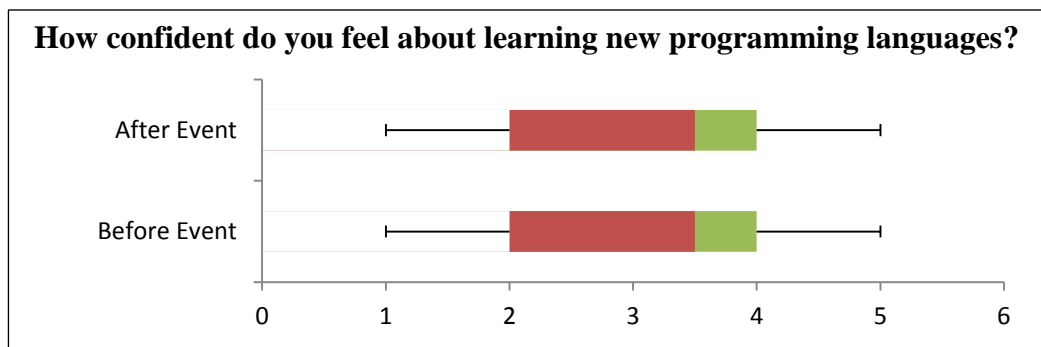


Figure 36: Box plot for the “How confident do you feel about learning new programming languages” question

On a similar theme students were also asked how confident they are using any programming language. Results are shown in Figure 37 and Figure 38.

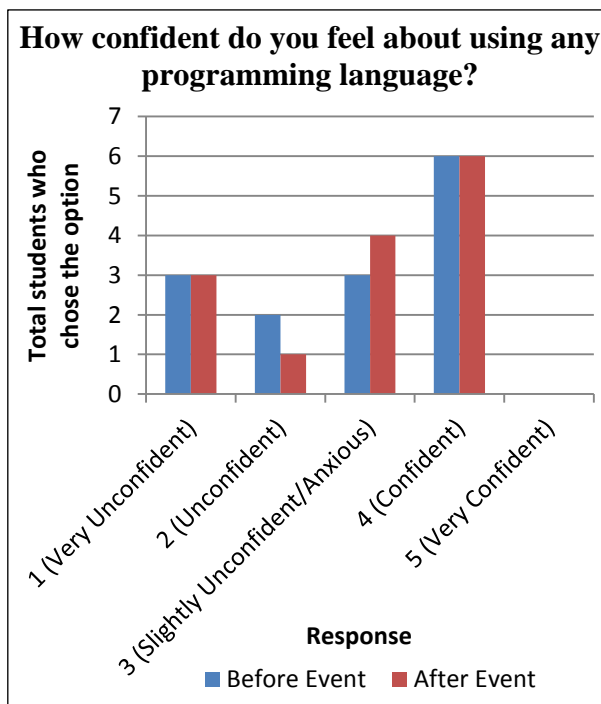


Figure 37: Students confidence using any programming language

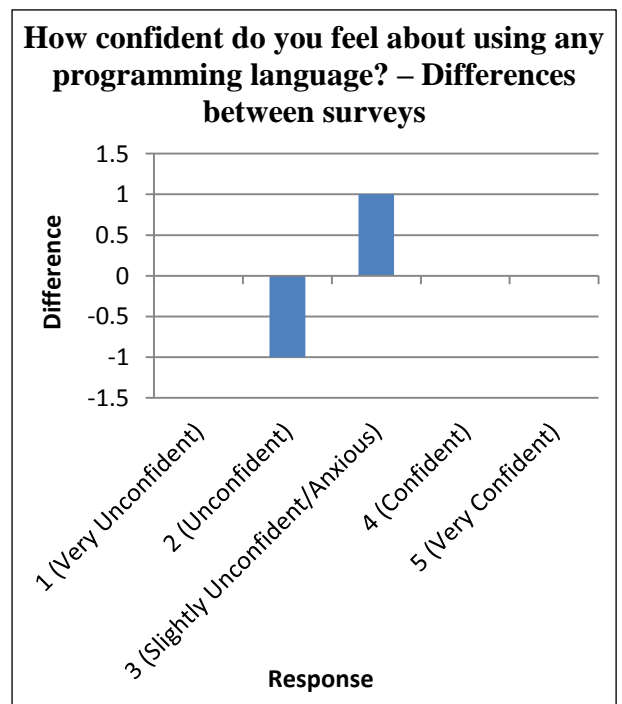


Figure 38: Students confidence using any programming language – Differences between surveys



In both surveys the majority of results are ‘confident’ or ‘slightly unconfident/anxious’. The event improved confidence slightly with an ‘unconfident’ response becoming ‘slightly unconfident/anxious’<sup>82</sup>. The responses were 42.86% positive, 21.43% slightly negative and 35.71% negative before the event and 42.86% positive, 28.57% slightly negative and 28.57% negative after the event.

Despite the differences in responses the averages and quartiles for the surveys (Table 21 and Table 22, and Figure 39) are the same with the exception of a 0.25 increase between the first quartiles and a 0.07 increase between the mean averages. The differences between surveys are not statistically significant (U= 95.5, Z= -0.0919, P= 0.92828).

Table 21: Quartiles for the “How confident do you feel about using any programming language” question

	Before Event	After Event
<b>Minimum</b>	1	1
<b>Quartile 1</b>	2	2.25
<b>Median (Q2)</b>	3	3
<b>Quartile 3</b>	4	4
<b>Maximum</b>	4	4

Table 22: Averages for the “How confident do you feel about using any programming language” question

	Before Event	After Event	Difference
<b>Mean</b>	2.86	2.93	0.07
<b>Median</b>	3	3	0.00
<b>Mode</b>	4	4	0

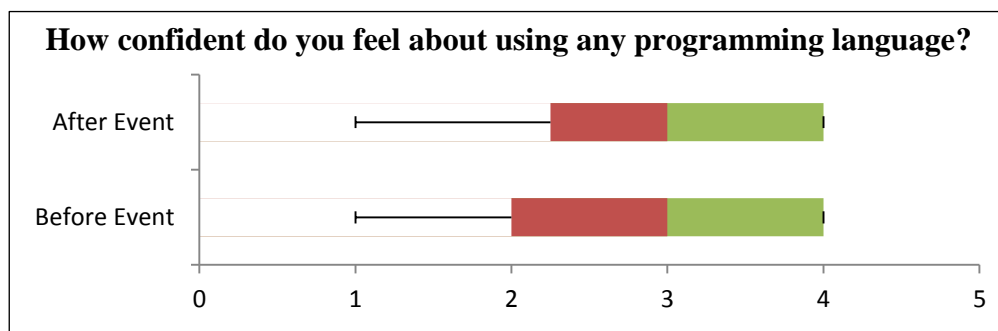


Figure 39: Box plot for the “How confident do you feel about using any programming language” question

These results suggest students are more confident learning new programming languages rather than using any programming language, which is surprising as it requires similar skills. This could suggest that students do not see how skills can be transferred between programming languages, how programming concepts are the same and many languages have similar syntax. Alternatively it could mean they find programming lessons easy so learning

<sup>82</sup> Although it is highly likely this is one person changing their response this cannot be identified due to anonymous results. It could be that multiple people changed results and the totals show just the one change.

new languages is considered easy but using languages without any instruction would be difficult so they don't feel confident just using any language.

#### 7.4. Rating skills

Students were also asked to rate their computing skills<sup>83</sup> (this uses a 1 to 5 scale where 1 = poor and 5 = excellent) and the results are in Figure 40 and Figure 41.

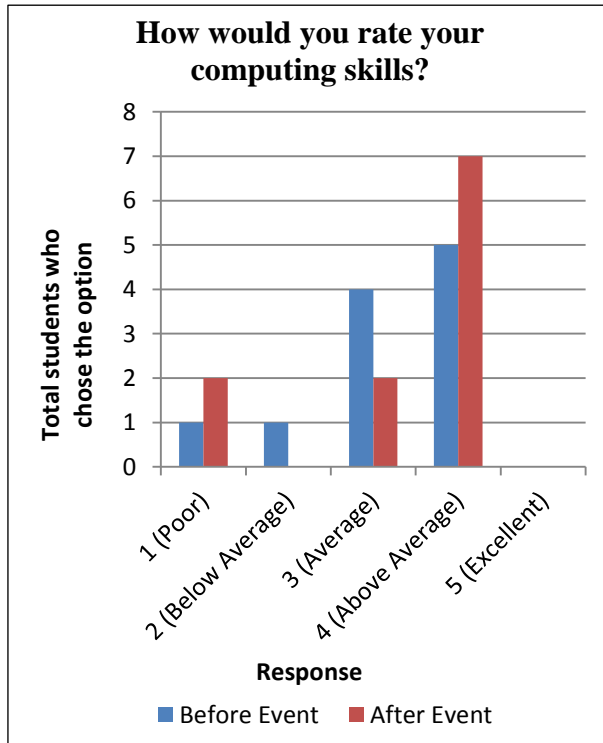


Figure 40: Students rating of their computing skills

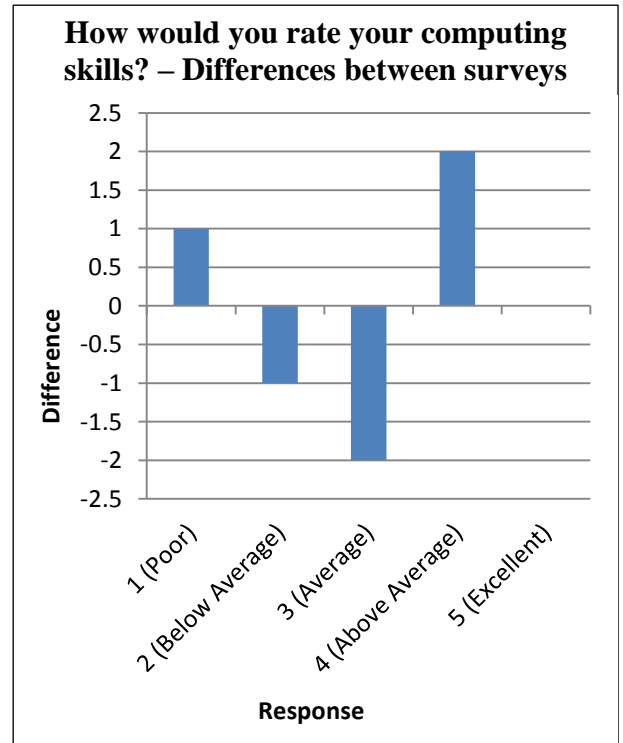


Figure 41: Students rating of their computing skills – Differences between surveys

The majority of students rated their skills as average or above average and the event increased the 'above average' responses; however there is a small increase in students considering their skills as poor. Responses<sup>84</sup> were 45.45% positive, 36.36% neutral and 18.18% negative for the before event survey and 63.63% positive, 18.18% neutral and 18.18% negative.

The averages and quartiles for the surveys (Table 23 and Table 24, and Figure 42) shows the results between surveys are similar with the exception of the median average increasing by 1 and the mean average increasing by 0.09 confirming an overall increase in students' ratings of their computing skills. The differences between surveys are not statistically significant (U= 52.5, Z= -0.4925, P= 0.62414).

<sup>83</sup> 3 students did not answer this question so there are 11 total responses rather than the usual 14.

<sup>84</sup> Above average and excellent are considered as positive responses, average is a neutral response and below average and poor are negative responses.

Table 23: Quartiles for the “How would you rate your computing skills” question

	Before Event	After Event
<b>Minimum</b>	1	1
<b>Quartile 1</b>	3	3
<b>Median (Q2)</b>	3	4
<b>Quartile 3</b>	4	4
<b>Maximum</b>	4	4

Table 24: Averages for the “How would you rate your computing skills” question

	Before Event	After Event	Difference
<b>Mean</b>	3.18	3.27	0.09
<b>Median</b>	3	4	1.00
<b>Mode</b>	4	4	0

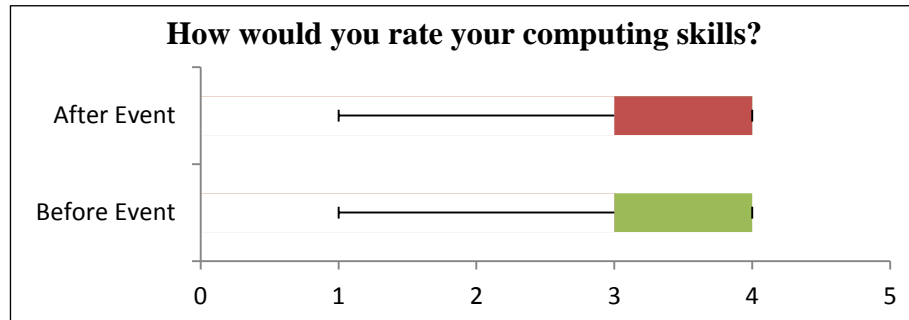


Figure 42: Box plot for the “How would you rate your computing skills” question

Similarly the students were asked to rate their programming skills<sup>85</sup> and the results are shown in Figure 43 and Figure 44.

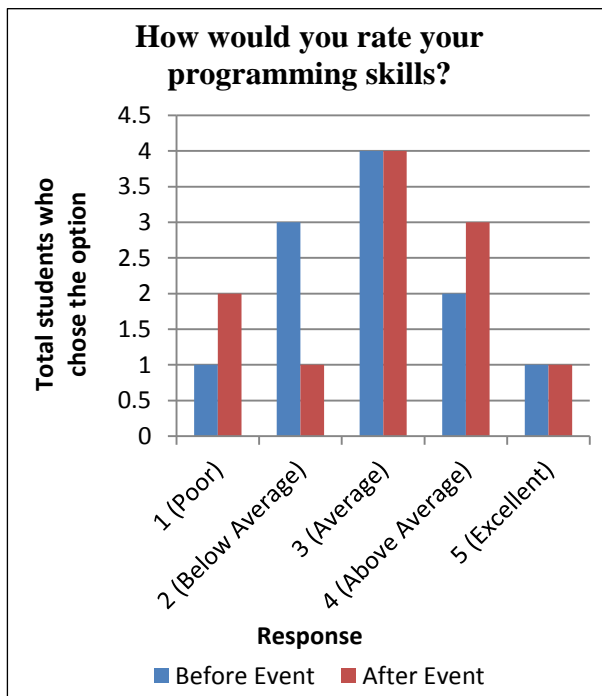


Figure 43: Students rating of their programming skills

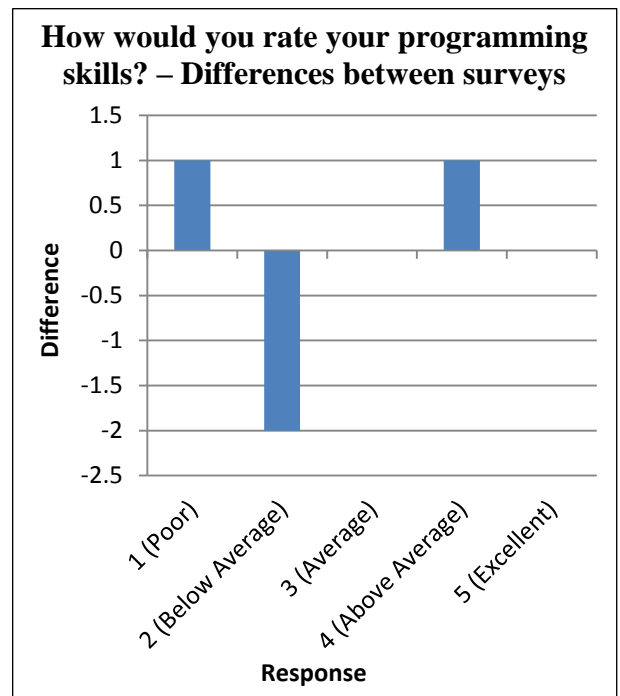


Figure 44: Students rating of their programming skills – Differences between surveys

<sup>85</sup> 3 students did not answer this question so there are 11 total responses rather than the usual 14.

The majority of students rated their programming skills as average or above and the event increased the ‘above average’ responses and reduced the ‘below average’ responses. However there is a slight increase in students considering their skills as poor. Responses were 27.27% positive, 36.36% neutral and 36.36% negative before the event and 36.36% positive, 36.36% neutral and 27.27% negative after the event.

The averages and quartiles for the surveys (Table 25 and Table 26, and Figure 45) also show the improved student ratings with increases for quartiles 1 and 3 and the mean average. The differences between surveys are not statistically significant ( $U= 56, Z= -0.2627, P= 0.79486$ ).

Table 25: Quartiles for the “How would you rate your programming skills” question

	Before Event	After Event
<b>Minimum</b>	1	1
<b>Quartile 1</b>	2	2.5
<b>Median (Q2)</b>	3	3
<b>Quartile 3</b>	3.5	4
<b>Maximum</b>	5	5

Table 26: Averages for the “How would you rate your programming skills” question

	Before Event	After Event	Difference
<b>Mean</b>	2.91	3.00	0.09
<b>Median</b>	3	3	0.00
<b>Mode</b>	3	3	0

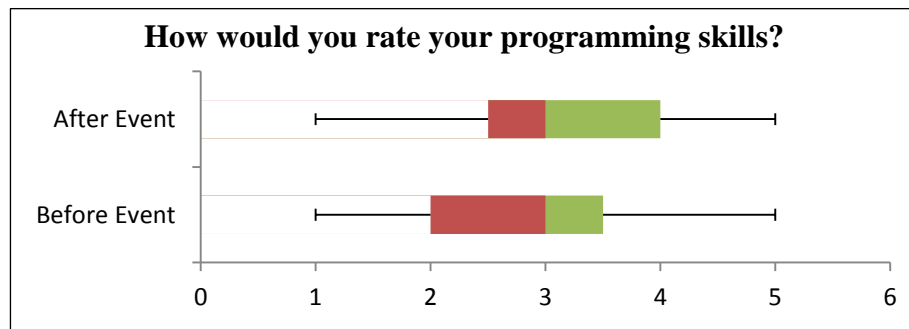


Figure 45: Box plot for the “How would you rate your programming skills” question

## 7.5. Summary

There are various observations that can be found from these results which can provide an insight into students' perceptions and understanding of computing and whether the case study's aims and objectives were met.

### Studying computing

- The majority of responses were positive (64.29%<sup>86</sup>) about considering studying ICT or Computing as a GCSE subject or an equivalent and the event slightly improved these opinions.
- The majority of responses were positive (42.86%) or neutral (28.57%) for considering AS/A level Computing or a college computing course and the event improved these opinions.
- Similarly most students' considerations of computing at university were positive (57.14%), although this slightly decreased after the event<sup>87</sup>.

### Career ambitions

- Before the event responses were slightly negative overall for considering getting a computing job and the event improved opinions to an equal split between positive and negative responses.
- 70% of respondents have considered working in the computing industry. This result conflicts with the previous question which showed only 50% of students considered getting a computing job<sup>88</sup>.
- The most popular sector to work in is game development which echoes the finding by Carter (2006).
- Students' main motivations for working in the computing industry are because it is seen as cool and has good job prospects.

---

<sup>86</sup> As explained above if you consider possibly as positive this becomes 85.72%.

<sup>87</sup> This may indicate that some students have reconsidered university as a suitable choice for them as a result of the event. Perhaps the event was too complex so they are less certain they have the skills for university or perhaps the event was too confusing or they didn't enjoy it and it made them re-evaluate the value of university.

<sup>88</sup> Perhaps the 2 questions although similar confused the students or they do not see having a computing job as the same as working in the computing industry.

## Computing Skills

- There is a reasonable amount of confidence for explaining ‘if’ statements which increases slightly as a result of the event.
- There was a reasonable amount of confident responses for explaining a loop prior to the event but overall confidence was low and after the event overall confidence reduced<sup>89</sup>.
- There were 41.67% positive responses for confidence explaining variables before the event which increased to 50% as a result of the event<sup>90</sup>.
- Confidence using Scratch is reasonably high with positive responses of 57.14% before the event rising to 64.29% after the event.
- Confidence in learning new programming languages was reasonably high with 50% answering with confident or very confident. However there was a large percentage of unconfident responses but after the event this reduced as overall confidence increased.
- There was a reasonable amount of confidence in using any programming language which increased slightly as a result of the event.

## Rating skills

- The majority of students rated their computing skills as average or above average and the event increased the ‘above average’ responses.
- The majority of students rated their programming skills as average or above and the event increased the ‘above average’ responses and reduced the ‘below average’ responses.

---

<sup>89</sup> This is surprising considering loops were covered in the tasks/examples as well as specifically being explained by the researcher.

<sup>90</sup> This is quite a small change and in general variable knowledge appears to be low. Unfortunately there wasn’t enough time to cover variables in the session so this increase is unlikely to be as a result of the event.

## 7.6. Limitations

Although many valuable results were acquired as a result of this event there were certain limitations which restricted the usefulness of the results and ability to draw strong conclusions.

It isn't possible to consider improvement in students views about working in the computing industry as the students didn't revisit these questions in the after event survey<sup>91</sup>.

Some students' responses appear to show little thought by responding with the same number for entire groups of questions or all the surveys questions<sup>92</sup> especially in the after event survey. It may be that they have equal views for these questions but it could also mean they are not paying much attention to their responses<sup>93 94</sup>.

2 students<sup>95</sup> ignored the variable question which could suggest they don't know what a variable is.

There is not much variation in the results for many questions and differences between surveys aren't significant enough to draw strong conclusions. This is probably because the pilot study was too small with only 14 students<sup>96</sup> and a larger survey would offer more opportunity for analysis.

Due to the need for event staff to provide the students with lots of help there was no time to make formal observations. However the researcher did discuss the event with the staff after the event and this feedback has been discussed previously in this dissertation. The researcher was also promised additional feedback would be provided which unfortunately was not received by the time this dissertation was published.

---

<sup>91</sup> This is probably because the same survey form was used for both surveys (for simplicity and to save time) and they didn't see the need to revisit these questions or perhaps their opinions didn't change.

<sup>92</sup> For example for one group all responses may be all 4, another all 3. One student just said 1 for all questions in the survey either indicating they were really confused about computing or (more likely) they were not interested in considering their responses or doing the survey.

<sup>93</sup> For example a common technique for quickly responding without much thought would be to choose the middle value (3).

<sup>94</sup> This lack of interest and attention could also be the reason for more negative responses as the students may not be giving any thought to their responses and just adding any number. They could even be adding more negative answers as they are annoyed to have to do another survey especially as it has almost all the same questions as the previous survey.

<sup>95</sup> 14.29% of the total students

<sup>96</sup> Some students ignored some questions further reducing the responses for those questions.

On analysing the responses it became clear that the ranges used could be improved to produce more useful results. Firstly there should be an equal amount of positive and negative results (or similar groupings) and no middle answer thus making it easy to define which answers belong to each group; this also helps remove the temptation for respondents to pick the middle answer rather than thinking about their responses. Secondly in the confidence range ‘slightly unconfident/anxious’ was not a suitable option as it can’t be properly defined as either a positive, negative or neutral response as it is slightly negative but not as clearly defined as the other negative responses; also if it was considered as negative it would create more negative responses than positive thus creating a negative bias.

### **7.7. Future improvements**

The pilot event showed the need for the following improvements if the event were to be repeated:

- Simplified content
- Survey response ranges with equal amounts of clearly defined positive and negative responses and no middle value
- Perhaps using additional modules for simplifying electronics or replacing the Raspberry Pi with another device (see appendix 16)
- A larger gap between surveys
- Perhaps make the event for older students such as year 10<sup>97</sup>
- Use of separate survey forms ideally online to enforce mandatory questions with a properly formatted paper-based survey as a backup/alternative

The attributes for comparing programming languages/tools/environments didn’t take into account level of course (such as introductory or advanced courses) or student’s ages or key stages. Therefore a future improvement could be to re-evaluate the languages/tools/environments to address this.

---

<sup>97</sup> These are 14-15 year olds who should have more computing knowledge and maturity and may be able to understand the tasks better. However an advantage of making the event for year 8 students was there limited experience so there is more opportunity to improve their opinions and skills.



## 7.8 Summary

Research shows there is a lack of interest in computing careers and Further and Higher Education courses. There are various possible reasons for this which highlighted the need to improve computing education in schools. A case study was created to help solve this by creating an event to improve students' perceptions and understanding of computing which also worked as a CPD opportunity for teachers. It was designed to show teachers how they can make fun, motivational, practical computing activities and demonstrate physical computing and electronics with programming.

Overall the students had positive perceptions and understanding of computing prior to the event and the event improved on these. However there were reductions in some areas such as the likelihood of studying computing at university and understanding of loops which suggests the event confused some students and they reconsidered their future study options. The students appeared to enjoy the event and found it fun and motivational which is supported by the increases in the survey results. It provided an excellent opportunity for students and teachers to improve their computing knowledge, opinions and confidence. The examples were practical and showed computing and programming's link with electronics and some had real-world relevance<sup>98</sup>. It provided teachers with ideas on interactive teaching methods and teaching practical computing. Although there were some slight decreases in some opinions and confidence the event has met its aims and objectives.

---

<sup>98</sup> For example the traffic light example showed the logic behind traffic lights. However it wasn't possible to cover all the planned examples.

## 8. Conclusions

This study investigated how computing is taught in education and ways to enhance students' perceptions and understanding of computing. It aimed to answer the research question: Can interactive teaching methods enhance students' perceptions and understanding of computing and increase their computing knowledge?

The study found there are clearly many problems contributing to overall poor perceptions and understanding of computing especially when students consider Further and Higher Education options and careers. One of the main problems is how computing is taught with poor quality computing education in schools, outdated university courses and so forth. In addition there are many misconceptions of what computing education and careers involve. Research showed how this contributed to low enrolment figures for computing courses at university and recruitment problems in industry in both the UK and USA.

A case study of a university outreach event for secondary school students was created to see whether interactive teaching methods can enhance students' perceptions and understanding of computing. This focussed around physical computing and was designed to make computing fun, motivational and relevant, and to provide examples of real-world applications. Suitable microcontrollers and microcomputers were evaluated which can show how computers work and how electronics and robotics can be used with computers as inputs of a program and for outputs as a result of coding, such as controlling lights, sensors, motors and so forth. Also programming languages/tools/environments that are commonly used in education were evaluated. A Raspberry Pi microcomputer was chosen to make use of its GPIO pins to control electronics and Scratch was chosen as the programming language/tool/environment due to its ability to simplify programming, and to interact with the GPIO pins on a Raspberry Pi.

The pilot event/case study appears to have improved the students' perceptions and understanding of computing but the sample size was small and there was little variation in responses, so it is not possible to create definitive conclusions without conducting a larger study. However it does indicate that interactive teaching methods enhance computing education and physical computing with electronics can enhance lessons. It shows the relevance of computing with examples of real-world applications and that it can be fun and motivational. With a few minor adjustments it can provide an excellent basis for a larger

study and if results remain positive can provide teachers with a proven way of enhancing computing lessons via interactive teaching methods.

These findings are similar to those found by the studies discussed in the literature review and they complement each other as useful ways to enhance students' interest in computing and their computing skills. The recurring theme across the research is that computing has a poor reputation and computing education requires rethinking, modernising and made more relevant to today's world. The strategies in these research projects have proven to help reduce these problems and other institutions would benefit from using them.

## 9. Recommendations and further work

To develop this work further it would be advisable to repeat the case study with a few improvements as discussed in section [7.7](#).

By repeating the event with more students either as a large group or multiple small groups (recommended<sup>99</sup>) would offer more opportunity for analysis and to validate if the findings from the pilot event/case study are accurate.

The case study showed that despite explaining programming concepts to students some students' confidence and understanding of them reduces; for example there was an overall reduction in students confidence in explaining loops and other questions where confidence reduced from 'unconfident' to 'very unconfident'. Therefore research could be conducted on why and how students' confidence in understanding a concept can reduce after it has been explained to them.

This work could form the basis for a much larger study such as a PhD<sup>100</sup> or EdD<sup>101</sup> investigating one or more of the following:

- Evaluating the case study against other methods for enhancing computing education.
- Investigate how computing is taught in feeder schools to secondary schools and look for areas to improve it so that all schools teach computing to the same level. If some schools teach little computing content and some a considerable amount then it creates an imbalance in secondary school classes. Recommendations could also be made for all feeder schools to teach the same content and thus secondary schools know they don't have to repeat this content; for example if all schools taught programming with Scratch then secondary schools will not need to start with introductory Scratch lessons.
- Develop a course for teachers to enhance their computing knowledge and computing teaching.

---

<sup>99</sup> It would be more difficult to run the event with a larger group and therefore it is recommended and much more simple and manageable to run the event for multiple small groups instead.

<sup>100</sup> Doctorate in Philosophy

<sup>101</sup> Doctorate in Education

- Investigate how computing is taught in other countries and evaluate teaching methods, curriculums and so forth to make recommendations on ways to improve global computing education.

Another possible area to cover would be to review learning resources and useful websites available to teachers; appendix 18 contains a list of learning resources and links as a basis for this.

## 10. References

- Alice, 2013. *Alice*. Carnegie Mellon University. Available from: <http://www.alice.org> [Accessed 13 June 2013].
- Albinson, P., 2013. A review into the factors affecting declines in undergraduate Computer Science enrolments and approaches for solving this problem. In: Davies, P., ed. *Proceedings of the Inspire DEC Student Conference 2013*, 31st May 2013 Poole. Poole: Bournemouth University.
- Arduino, 2013. *Arduino*. Arduino. Available from: <http://www.arduino.cc/> [Accessed 3 July 2013].
- BCS, 2013a. *BCS, The Chartered Institute for IT in association with the Computing At School group Consultation Response to: Reform of the National Curriculum in England*. Swindon: BCS. Available from: <http://academy.bcs.org/sites/academy.bcs.org/files/BCS%20National%20Curriculum%20Response%20-%20April%202013.pdf> [Accessed 4 July 2013].
- BCS, 2013b. *Teaching scholarships | BCS Academy of Computing*. Swindon: BCS. Available from: <http://academy.bcs.org/scholarships> [Accessed 12 August 2013].
- BeagleBoard, 2013. *BeagleBone*. The BeagleBoard.org Foundation. Available from: <http://beagleboard.org/Products/BeagleBone> [Accessed 3 July 2013].
- BlueJ, 2013. *BlueJ*. BlueJ. Available from: <http://www.bluej.org> [Accessed 13 June 2013].
- BU, 2009. *Research Ethics Code of Practice*. Poole: Bournemouth University. Available from: <http://portal.bournemouth.ac.uk/sites/Policies%20Procedures%20and%20Regulations/Shared%20Documents/Research%20Ethics%20Code%20of%20Practice%20Sept%202009.pdf> [Accessed 09 July 2013].
- Carter, L. 2006. Why Students with an Apparent Aptitude for Computer Science Don't Choose to Major in Computer Science. In: *Proceedings of the 37th SIGCSE technical symposium on Computer Science education, SIGCSE '06*, 1-5 March 2006, Houston, Texas, New York: ACM, 27-31. Available from: <http://dl.acm.org/citation.cfm?id=1121352> [Accessed 17 May 2013].
- Converting to Computer Science, 2013. *How should teachers be taught to program?* Converting to Computer Science. Available from: <http://www.convertingtocomputing.co.uk/teaching/how-should-teachers-be-taught-to-program/> [Accessed 21 June 2013].
- Cooper, S., Dann, W. and Harrison, J., 2010. A K-12 College Partnership. In: *Proceedings of the 41st ACM technical symposium on Computer science education, SIGCSE '10*, 10-13 March 2010 Milwaukee. New York: ACM, 320 – 324. Available from: <http://dl.acm.org/citation.cfm?id=1734371> [Accessed 17 May 2013].
- CRA, 2013. *CRA Taulbee Survey*. The Computing Research Association. Available from: <http://www.cra.org/resources/taulbee/> [Accessed 9 August 2013].
- Cymplecy, 2013. *Scratch GPIO Version 2 – Introduction for Beginners*. Cymplecy. Available from: <http://cymplecy.wordpress.com/2013/04/22/scratch-gpio-version-2-introduction-for-beginners> [Accessed 16 August 2013].
- Dawson, C., 2007. *Practical Guide to Research Methods: A User-Friendly Manual for Mastering Research Techniques and Projects*. 3rd Ed. Oxford: How to Books.
- De Leeuw, E.D., 2011. Improving Data Quality when Surveying Children and Adolescents: Cognitive and Social Development and its Role in Questionnaire Construction and Pretesting. In: *Proceedings of Annual Meeting of the Academy of Finland: Research Programs Public Health Challenges and Health and Welfare of Children and Young People*, 10-12 May 2011 Naantali, Finland, Helsinki: Academy of Finland. Available from: [http://www.aka.fi/Tiedostot/Tiedostot/LAPSET/Presentations%20of%20the%20annual%20seminar%2010-12%20May%202011/Surveying%20Children%20and%20adolescents\\_de%20Leeuw.pdf](http://www.aka.fi/Tiedostot/Tiedostot/LAPSET/Presentations%20of%20the%20annual%20seminar%2010-12%20May%202011/Surveying%20Children%20and%20adolescents_de%20Leeuw.pdf) [Accessed 16 July 2013].
- DfE, 2013a. *Computing: Programmes of study for Key Stages 1-4*. Department for Education. Available from: [http://computingatschool.org.uk/data/uploads/computing-04-02-13\\_001.pdf](http://computingatschool.org.uk/data/uploads/computing-04-02-13_001.pdf) [Accessed 06 June 2013].
- DfE, 2013b. *Reform of the National Curriculum in England*. Department for Education. Available from: <http://media.education.gov.uk/assets/files/pdf/n/national%20curriculum%20consultation%20document%20070213.pdf> [Accessed 20 June 2013].
- DfE, 2013c. *Computer science to be included in the EBacc*. Department for Education. Available from: <https://www.gov.uk/government/news/computer-science-to-be-included-in-the-ebacc> [Accessed 07 June 2013].
- Greenfoot, 2013. *Greenfoot*. Greenfoot. Available from: <http://www.greenfoot.org> [Accessed 13 June 2013].
- Grover, S., 2013. *OPINION: Learning to Code Isn't Enough*. edSurge. Available from: <https://www.edsurge.com/n/2013-05-28-opinion-learning-to-code-isn-t-enough> [Accessed 21 June 2013].
- HESA, 2013. *Joint Academic Coding System (JACS) Version 3.0*. Cheltenham: Higher Education Statistics Agency. Available from: <http://www.hesa.ac.uk/content/view/1776/649/> [Accessed 15 August 2013].
- Java, 2013. *Java*. Oracle. Available at <http://www.java.com> [Accessed 13 June 2013].
- Maker Media, 2013. *Arduino Uno vs BeagleBone vs Raspberry Pi*. Maker Media. Available from: <http://makezine.com/2013/04/15/arduino-uno-vs-beaglebone-vs-raspberry-pi> [Accessed 3 July 2013]

Microsoft, 2012a. *C# Language Specification 5.0*. Microsoft Corporation. Available from: <http://www.microsoft.com/en-us/download/details.aspx?id=7029> [Accessed 13 June 2013].

Microsoft, 2012b. *What are the 14 Keywords of Small Basic?* Microsoft Corporation. Available from: <http://blogs.msdn.com/b/smallbasic/archive/2012/10/08/what-are-the-14-keywords-of-small-basic.aspx> [Accessed 13 June 2013].

Microsoft, 2013a. *Microsoft Small Basic*. Microsoft Corporation. Available from: <http://smallbasic.com/> [Accessed 13 June 2013].

Microsoft, 2013b. *FAQ*. Microsoft Corporation. Available from: <http://smallbasic.com/faq.aspx> [Accessed 13 June 2013].

Microsoft, 2013c. *Microsoft Small Basic: An introduction to Programming*. Microsoft Corporation. Available from: <http://download.microsoft.com/download/9/0/6/90616372-C4BF-4628-BC82-BD709635220D/Introducing%20Small%20Basic.pdf> [Accessed 13 June 2013].

Microsoft, 2013d. *Small Basic Curriculum*. Microsoft Corporation. Available from: <http://social.technet.microsoft.com/wiki/contents/articles/16299.small-basic-curriculum.aspx> [Accessed 13 June 2013].

Microsoft, 2013e. *Small Basic E-Books*. Microsoft Corporation. Available from: <http://social.technet.microsoft.com/wiki/contents/articles/16386.small-basic-e-books.aspx> [Accessed 15 August 2013].

Morelli, R., de Lanerolle, T., Lake, P., Limardo, N., Tamotsu, E., and Uche, C., 2010. *Can Android App Inventor Bring Computational Thinking to K-12?* The Humanitarian FOSS Project. Available from: [http://hfoss.org/uploads/docs/appinventor\\_manuscript.pdf](http://hfoss.org/uploads/docs/appinventor_manuscript.pdf) [Accessed 21 February 2013].

National Children's Bureau, 2011. *Guidelines for Research with Children and Young People*. National Children's Bureau. Available from: [http://www.ncb.org.uk/media/434791/guidelines\\_for\\_research\\_with\\_cyp.pdf](http://www.ncb.org.uk/media/434791/guidelines_for_research_with_cyp.pdf) [Accessed 9 July 2013].

Pfleeger, S.L., 2001. *Software Engineering: Theory and Practice*. 2nd ed. Upper Saddle River: Prentice Hall.

Purewal Jr., T.S., 2010. Social Networking: The New Computer Fluency? In: *Proceedings of the 41st ACM technical symposium on Computer science education*, SIGCSE '10, 10-13 March 2010 Milwaukee. New York: ACM, 112-116. Available from: <http://dl.acm.org/citation.cfm?id=1734301> [Accessed 17 May 2013].

Punch, S., 2002. *Research with Children: The Same or Different from Research with Adults?* SAGE Publications. Available from: <http://chd.sagepub.com/content/9/3/321> [Accessed 16 July 2013].

PythonCode.co.uk, 2013. *Debug*. PythonCode.co.uk. Available from: <http://www.pythoncode.co.uk/debug> [Accessed 15 August 2013].

Raspberry Pi, 2013. *Raspberry Pi*. Cambridge: Raspberry Pi. Available from: <http://www.raspberrypi.org/> [Accessed 3 July 2013].

Roamer, 2013. *Roamer*. Roamer. Available from: <http://www.roamer-robot.com/public/> [Accessed 15 August 2013].

ScratchJr, 2013. *ScratchJr*. ScratchJr. Available from: <http://ase.tufts.edu/DevTech/ScratchJr/ScratchJrHome.asp> [Accessed 12 August 2013].

The Royal Society, 2012. *Shut down or restart? The way forward for computing in UK schools*. London: The Royal Society. Available from: <http://royalsociety.org/education/policy/computing-in-schools/report/> [Accessed 6 June 2013].

UCAS, 2013a. *UCAS*. Cheltenham: UCAS. Available from: <http://ucas.com/> [Accessed 1 August 2013].

UCAS, 2013b. *Annual data files*. Cheltenham: UCAS. Available from: <http://ucas.com/sites/default/files/annual-data.zip> [Accessed 1 August 2013].

UCAS, 2013c. *Data resources*. Cheltenham: UCAS. Available from: <http://ucas.com/data-analysis/data-resources> [Accessed 1 August 2013].

Wing, J. M., 2006. Computational Thinking. *Communications of the ACM*, 49 (3), 33-35. Available from: <http://dl.acm.org/citation.cfm?id=1118215> [Accessed 21 June 2013].

Yin, R.K., 2003. *Case Study Research: Design and Methods*. 3rd Ed. Thousand Oaks, California: Sage.

## 11. Appendices

### Appendix 1 – Literature review

# A review into the factors affecting declines in undergraduate Computer Science enrolments and approaches for solving this problem

Paul Albinson BSc (Hons), FdSc, MBCS  
Design, Engineering and Computing  
Bournemouth University  
Poole, England  
research@paulalbinson.info

**Abstract—** There has been a noticeable drop in enrolments in Computer Science (CS) courses and interest in CS careers in recent years while demand for CS skills is increasing dramatically. Not only are such skills useful for CS jobs but for all forms of business and to some extent personal lives as Information Technology (IT) is becoming ubiquitous and essential for most aspects of modern life. Therefore it is essential to address this lack of interest and skills to not only fill the demand for CS employees but to provide students with the CS skills they need for modern life especially for improving their employability and skills for further study.

This report looks at possible reasons for the lack of interest in CS and different approaches used to enhance CS education and improve the appeal of CS.

**Index Terms - Improving Computer Science Education; CS; CS0; CS1; ICT to Computer Science; Decreasing Computer Science Enrolments; Pedagogy; Motivation; Engagement.**

#### I. INTRODUCTION

There has been a noticeable drop in enrolments in Computer Science (CS) courses and interest in CS careers in recent years (approximately since 2000) while demand for CS skills is increasing dramatically. Not only are such skills useful for CS jobs but for all forms of business and to some extent personal lives, as Information Technology (IT) is becoming ubiquitous and essential for most aspects of modern life. Learning CS can also assist with learning other subjects as, for example, programming can teach: design skills (from ideas to finished products), problem solving and perseverance (identifying and fixing faulty code) and team work/collaboration skills. In addition having a solid understanding of CS will assist with the use of applications and processes in work and education such as secretarial skills, accounting skills, operating manufacturing design and production tools etc. Therefore it is essential to address this lack of interest and skills to not only fill the demand for CS

employees but to provide students with the CS skills they need for modern life, especially for improving their employability and skills for further study both formal and self-study.

One of the main theories for the unpopularity of CS is due to the way computing is introduced in schools, leading to a poor perception and understanding of what CS is. Computing education in schools typically focuses around Information Communications Technology (ICT) which is how to use computers and typically ignores CS, which is how and why computers work to provide a fuller understanding of computing and its value and potential.

The need to improve computing education has been recognised by governments, industry, professional bodies and education providers and has led to curriculums and guidelines being improved to provide a higher quality of computing education.

There are many tools and courses being created or improved to make CS easier to understand, improve engagement and motivation and to show the relevance of CS. There are signs that these approaches are effective and enrolment numbers are slowly increasing. However more work will be required to maintain this growth and interest such as ensuring the content remains relevant.

In addition to improving CS courses there have also been many initiatives to introduce what CS involves and ideally motivate students to consider a CS course and/or career. In the USA college/university<sup>102</sup> students choose a subject to specialise in, known as a major, and they can also study elective subjects in other areas which are known as non-majors. These non-major courses may be studied prior to major courses as an introduction to a subject as either a prerequisite to the major course (either as a course or university requirement) or simply to help students decide if the subject is of significant interest to study as a major. Most papers reviewed are from the USA and focus on making CS more interesting via either a non-major course, with the aim

---

<sup>102</sup> In the United States of America the term college is used to refer to part of a university (similar to a school in the UK university system) or as a stand-alone higher education institution. High Schools are the USA equivalent to the UK college system.



of encouraging students to consider a CS major, or assist those progressing onto a CS major, or by improving major courses to enhance interest in CS and improve retention rates and students grades.

Other ways of encouraging students to consider a CS course and career as well as improving their CS skills are summer schools, introductory courses, bridging courses, and school visits/outreach projects.

The remainder of this report looks at these topics in more detail to discover the reasons for the lack of interest in CS and different approaches used to enhance CS education and improve the appeal of CS.

## II. THE ENROLMENT CRISIS

Many papers refer to decreasing enrolments in CS courses which started around 2000. Most papers refer to the findings of the current editions of the Computer Research Association's Taulbee survey<sup>103</sup>. Morelli et al. (2010), Cooper et al. (2010) and Purewal Jr. (2010) consider the results from the 2007-2008 Taulbee survey (Zweben 2009) which shows that from 1995 – 2000 there were significant increases in new CS/CE<sup>104</sup> undergraduate majors<sup>105</sup>. There were significant decreases from 2000 to 2007 with a slight increase in 2008 (see figure 1). The survey also shows that the amount of bachelor's degrees produced follows a similar but slightly smoother pattern, with increases until a dip in 2003, followed by decreases from 2004 and a projected increase in 2009 (see figure 2).

However as mentioned in the survey report and by Cooper et al. (2010) the slight increase in enrolments in 2008 is probably influenced by a change in the way data was collected to include a broader range of CS/CE courses.

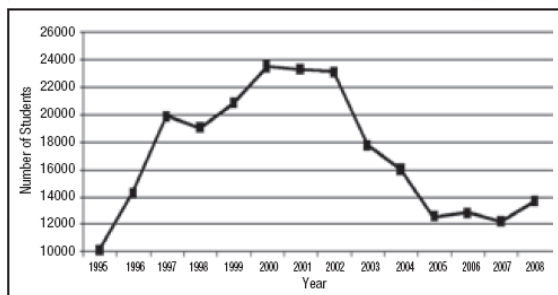


Figure 1 – Taulbee Survey: Newly Declared CS/CE Undergraduate Majors (Zweben 2009)

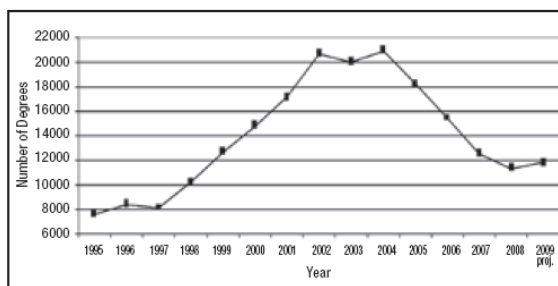


Figure 2 – Taulbee Survey: CS & CE Bachelor's degree production (Zweben 2009)

Uludag et al. (2011) and Wolber (2011) consider the 2008-2009 Taulbee survey results (Zweben 2010) which shows a small continued increase in CS majors (see figure 3) yet it is still nearly 50% lower than in 2000. However as Uludag et al. (2011) reports in the 2009 survey, the degree production figures continue to decrease; perhaps the increased enrolments aren't affecting this as the new students aren't ready to graduate. In addition the survey shows that the prediction of increased degree production in 2009 was incorrect as the figure decreased, they however predict an increase in 2010 (see figure 4).

These results along with other data prompted many institutions to work on improving the appeal of CS. When we look at the latest Taulbee survey (Zweben 2013) we see a small decrease and stagnation in undergraduate enrolments between 2009 and 2011 and a massive increase in 2012 (see figure 5); in addition since 2009 there has been a steady increase in bachelor degree production (see figure 6) suggesting these initiatives are effective.

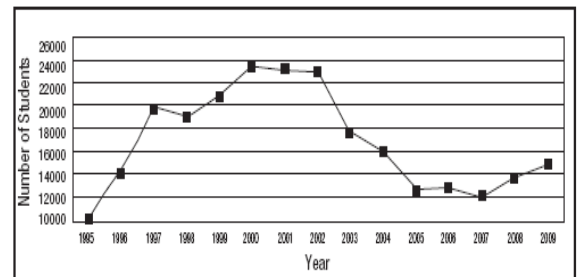


Figure 3 – Taulbee Survey: Newly Declared CS/CE Undergraduate Majors (Zweben 2010)

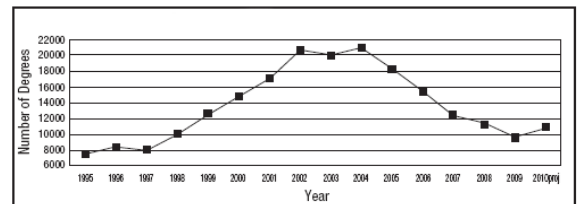


Figure 4 – Taulbee Survey: CS & CE Bachelor's degree production (Zweben 2010)

<sup>103</sup> <http://www.cra.org/resources/taulbee/>

<sup>104</sup> Computing Engineering

<sup>105</sup> There was however a slight decrease in 1998.

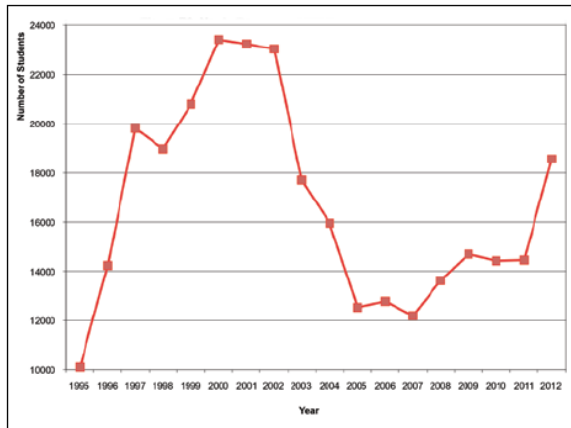


Figure 5 – Taulbee Survey: Newly Declared CS/CE Undergraduate Majors (Zweben 2013)

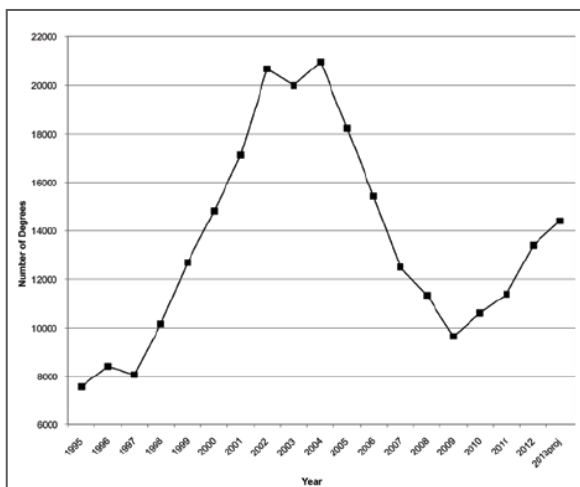


Figure 6 – Taulbee Survey: CS & CE Bachelor's degree production (Zweben 2013)

Other papers also discuss other data around enrolments which show similar results. Malan (2010) explains the enrolments for Harvard's CS50 CS course which has similar levels to the national figures shown in the Taulbee survey with enrolments rising from 1993, peaking in 1996 before reducing in subsequent years with sudden massive drops in 2001 and 2002; these rises and falls correspond to the start and end of the dot-com boom/bubble when a lot of money was made and subsequently lost with internet start-ups, hence interest in CS was sparked and lost accordingly. Their enrolment rates slowly increased until 2006 when they improved the course content to being more relevant and appealing resulting in subsequent sharp enrolment increases (see figure 7). Sahami et al. (2010) also reports similar drops in enrolments at Stanford between 2001 and 2006.

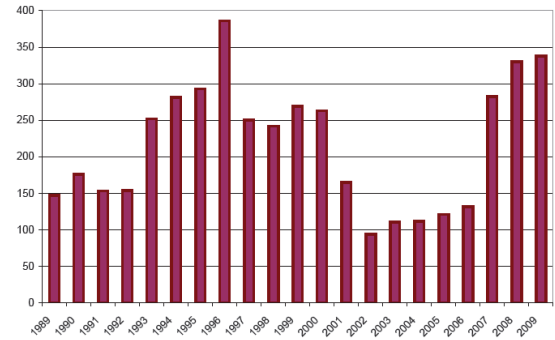


Figure 7 – Enrolments in Harvard's CS50 course (Malan 2010)

The lack of interest in CS is a potential crisis as there is an ever increasing demand for CS skills not only for CS jobs but for use in most jobs, due to IT being essential for the running of modern businesses. Egan (2010) discusses the problem and how U.S. Department of Labor (2007) surveys suggest that jobs in the computing industry will increase dramatically by 489,000 jobs between 2006 and 2016 while CS graduate rates remain low.

### III. POSSIBLE REASONS FOR LACK OF INTEREST IN CS

#### A. Outsourcing

Sahami (2007) and Sahami et al. (2010) speculate that the health of the technology economy and increases in outsourcing jobs may discourage students from considering a CS career. However they note that a more detailed analysis of such factors by Aspray et al. (2006) shows how outsourcing hasn't resulted in a net loss due to an overall increase in IT jobs. Therefore any reduced enrolments would be due to a perception of reduced jobs rather than actual job reductions.

#### B. CS isn't cool

Various papers discuss how CS is often poorly perceived and understood and how courses should be modernised and portray the value, relevance and appeal of CS, ideally with real-world examples. Malan (2010) hypothesised that the problem with enrolment decreases in Harvard's CS50 introductory CS course for both majors and non-majors was due to the courses design and the students' perception of it. The design of the course was seen as a problem as the content may be seen as dated, especially with students being more aware of technology and having modern technology such as smartphones, laptops etc. They also assumed that the workload and perceived difficulty of the course is a reason for its unpopularity. They concluded that the course needs to be redesigned to include more modern content and make it more accessible, motivating and appealing to students. They ideally wanted to recreate the large increase in CS enrolments that external factors like the dot-com boom/bubble created, but with internal factors such as improved course content which will hopefully maintain interest longer. They reorganised the course structure,

modernised the content and where possible linked it to real-world problems/scenarios. For example starting with a simple “hello world” programming example<sup>106</sup> is not a very exciting/motivating first lesson for a modern course; when computing power was limited and less graphical this was fine but in the modern world it seems very dull. The new course has the same level of complexity and workload but is more accessible and friendly to make it less daunting/scary to encourage more students to realise that the course is suitable for them. This approach is vital when teaching non-majors as well as majors, as students will have varying pre-existing CS skills and experience, so content needs to be approachable yet significantly complex to accommodate varied skill levels. They found the improvements increased interest in CS and made the course more appealing and increased enrolments as well as enrolments of subsequent courses.

Kurkovsky (2007) also refers to there being many misconceptions about CS as their study showed students do not understand what CS is, feel it is more difficult than other subjects and often consider it as “nerdy” and “not cool”. It is particularly difficult to change these opinions of non-CS majors (students studying CS as a non-major course and are probably only studying an introduction to CS course as a requirement of their major course) as they probably have little interest in the subject. They explain how CS courses for non-majors are typically either computer literacy (how to use computers such as using office applications) which doesn’t include programming or a “CS 0” course (how computers work) which includes a comprehensive overview of CS and usually introduces programming. They also discuss the value of teaching programming to non-CS majors including research for and against the point. One approach mentioned is to highly tailor programming content around specific industries as proposed by Forte and Guzdial (2005) who also evaluate the value of programming for non-CS majors.

### C. Other reasons

Carter (2006) considers the reasons for why enrolments for CS majors are reducing across the USA and why students with an apparent aptitude for CS, such as high-school calculus and pre-calculus students, avoid CS as a major and whether reasons vary by gender. As with other studies they observed massive drops in CS major enrolments and considered similar hypotheses to explain this (outsourcing, the dot-com bust, negative perceptions of CS, lack of or incorrect information on what CS is, gender differences etc.). They also assume that high schools are not introducing CS to their students and they have little understanding of what CS is; from examining course catalogues they found there was little or no CS content. They surveyed high school calculus and pre-calculus students as maths success is typically a predictor of CS success, to establish possible reasons for why these ideal students aren’t enrolling and whether the reasons vary by

<sup>106</sup> Traditionally programming tutorials begin with how to create a program to write “Hello World” to the screen.

gender. The results confirmed that high-school students lack computing experience and do not fully understand what CS involves. The main reasons for avoiding a CS major were the same for both genders and were the misconception that CS involves working with a computer all day, or they had already chosen to study a different course. The main reasons for studying CS varied by gender; men state computer games as their main influence/interest whereas women saw CS skills as being useful for other fields. Potential earnings were not a consideration.

## IV. CS RATHER THAN ICT

One of the theories for why CS is misunderstood and unpopular is the way computing is introduced in schools. Traditionally Information and Communication Technology (ICT) is taught, which is how to use computers<sup>107</sup>, rather than CS which is “the study of the foundational principles and practices of computation and computational thinking, and their application in the design and development of computer systems” (Naace, ITTE, and the Computing at School Working Group 2012, p.1)<sup>108</sup>. This neglect of CS in computing introductions fails to explain the fundamental principles of computing and show the relevance of CS. It creates a poor impression of CS and can fail to motivate students to pursue further CS study and careers. This problem has been recognised by the UK government who are scrapping the ICT GCSE and are proposing a new computing curriculum and GCSE (Department for Education 2013). The curriculum is for key stages 1 – 4<sup>109</sup> and provides a more complete computing education and aims to provide a solid understanding of CS and ICT required for industry and further study. It includes fundamentals of CS, computational thinking and evaluating and using ICT.

A similar approach is being taken in the USA where a National Research Council review into IT literacy as requested by the National Science Foundation (Lin 2000) concluded that computer literacy (a.k.a. ICT) should be replaced by IT fluency (a.k.a. CS). They explain how as modern computing changes regularly, computer literacy skills (how to use current applications) soon become obsolete. However as IT fluency teaches computing fundamentals and principles it provides more flexible skills to expand knowledge and adapt to changes; for example a user may not completely understand a program but has the skills to learn it themselves.

Scott Hilberg and Meiselwitz (2008) explain how, due to the importance and prevalence of IT in modern life, it is essential for students to have IT/ICT fluency skills. However, despite growing up with modern IT there are concerns that students lack these skills; they reference previous research supporting this. They also

<sup>107</sup> Students are typically only taught how to use the Microsoft Office suite of office applications and similar basic computer uses such as web browsing.

<sup>108</sup> Naace, ITTE, and the Computing at School Working Group (2012) provide a more in-depth comparison of ICT and Computer Science.

<sup>109</sup> This is the entire UK school system from ages 5-16.

say how students' consider their ICT fluency skills as good (faculty and administrators commonly make similar assumptions) yet actual ICT results are typically lower. They investigated perceived knowledge via a survey and actual knowledge using an Educational Testing Service's ICT Literacy Assessment. Results show the mean score was 158.20 which is just over half the possible marks (53.79%) and shows that most students have poor ICT fluency skills. The majority of the students (73%) were overly confident of their ICT skills and achieved lower scores than their perceptions. Also those who overestimated their skills were more than double those who underestimated their skills (26%). The low ICT fluency skills observed are despite more than three quarters (79.8%) of undergraduates having had past ICT training which indicates current ICT training is not sufficient for teaching the required ICT fluency skills. They conclude that the ICT curriculum needs evaluating to ensure students have the required ICT fluency skills.

Dougherty (2003) also explains the need for students to be fluent with IT due to its importance and prominence in the modern world and because it is always changing. There have been previous attempts to teach the required IT skills for the workforce which initially started by concentrating on IT literacy. However literacy is not scalable enough to take into account the constantly changing nature of IT and training changed to focus on IT fluency. They then discuss and define IT fluency and reference related reports. They also explain how many colleges and universities have been creating computing courses for non-majors (with references to examples) and how the ACM/IEEE Computing Curricula 2001 (ACM/IEEE CS Joint Task Force on Computing Curricula 2001) identifies the need for IT fluency in CS courses. They then discuss the IT Fluency (ITF) Framework (Dougherty et al. 2002) which is "a case study template that can be used to design and implement a set of laboratory exercises in a field outside of computing with non-trivial usage of IT" along with how they used it within their "The World of Computing" course at Haverford College<sup>110</sup>. This was implemented as 1 day of IT fluency lessons based around an economics case study. They explain the day's assessments and a survey conducted to assess its effectiveness. They conclude that the day's lessons went well but they felt it would be more effective if they could expand it to at least 2 days to allow the addition of some brief examples and more time to absorb the content and clarify queries. Unfortunately only 10% of students managed to repeat the demonstration on their own and many of these needed significant help to achieve this. Student feedback was positive but students were confused by some of the survey questions so they couldn't draw solid conclusions from it. They feel it is worth repeating the use of the ITF framework but will make some minor

---

<sup>110</sup> It was seen as impractical to base an entire course on the ITF framework.

changes such as adding a second case study on psychology.

## V. GOVERNMENT AND INDUSTRY SUPPORT

The value of CS has been recognised by governments and industry; in addition to the aforementioned new computing curriculum and focus on CS rather than purely ICT there are many initiatives to improve CS teaching (including ICT content). These initiatives are supported by many schools, universities, governments and industry including BCS, Google, Microsoft, Facebook and many more. The Computing at School working group/initiative (CAS) brings together educators and industry to work on improving CS and share knowledge. CAS has worked with the BCS Academy to create the Network of Teaching Excellence in Computer Science. The network helps educate teachers to increase the level of CS teaching. It includes such initiatives as universities training school teachers so they can provide their students with the skills required for college and university CS courses. There are similar initiatives around the world such as the focus on IT fluency in the USA. Other examples are computer clubs and programs/applications to introduce children to CS (this is typically via programming) such as Code Club<sup>111</sup>, Code.org<sup>112</sup>, Google Computer Science for High School<sup>113</sup> and Google Summer of Code<sup>114</sup>. In addition there are plenty of other resources such as online courses like Coursera<sup>115</sup> and Khan Academy<sup>116</sup> designed to make learning accessible to all.

## VI. POSSIBLE SOLUTIONS

With a clear need to enhance students' perception of CS and create appealing CS courses, many different approaches/solutions have been investigated.

### A. Tailored courses

Forte and Guzdial (2005) explain how like many institutions Georgia Institute of Technology (Georgia Tech) requires majors and non-majors to study an introductory CS course and such courses have difficulty engaging non-CS students. As a possible solution they introduced two tailored introductory CS courses for non-majors (students interested in or majoring in certain non-CS areas) as an alternative to their traditional course "Introduction to Computing". The tailored courses are "Introduction to Computing for Engineers" (tailored for engineers) and "Introduction to Media Computation" (tailored for non-CS and non-engineering students). They hope by showing students how CS is relevant for their chosen industry they will see the value of CS and find it more understandable and interesting. Also the tailored approach creates a more balanced class of peers with similar skills, backgrounds and interests which helps

---

<sup>111</sup> <http://www.codeclub.org.uk/>

<sup>112</sup> <http://www.code.org/>

<sup>113</sup> <http://www.cs4hs.com/>

<sup>114</sup> <https://developers.google.com/open-source/soc/>

<sup>115</sup> <https://www.coursera.org/>

<sup>116</sup> <https://www.khanacademy.org/>

with students' comfort and confidence. They ensured the content and especially the chosen programming language was relevant to the audience and is useful in their chosen careers; for example Java is typically used by engineers. Also learning objectives and assessments need to be considered to take into account the new content and the audience. They found these new courses were much more effective than the traditional course with more students completing and passing the courses and they received more positive (and less negative) feedback with many students wishing to study another tailored course.

### *B. Improving and modernizing courses*

Sahami et al. (2010) explain how despite significant evolution of computing in the last 30 years the CS curricula hasn't adapted accordingly. With this in mind and a noticed reduction in CS enrolments, Stanford University redesigned its CS curriculum to modernise it. Their goals were: to add flexibility to adapt content to keep it relevant, include modern content and highlight future developments, emphasise the breadth of potential CS areas, provide options for exploring areas in depth, and show the diversity and multi-disciplinary nature of CS. The restructured curriculum contains:

- Core units provide a solid foundation for the course and cover CS fundamentals and principles along with topics to explain modern concepts which could form the basis for future computing developments.
- Depth concentration in a track area – Students can choose units in the area they wish to concentrate on/specialise in as well as related multi-disciplinary content.
- Elective units provide students with a choice of units designed to provide more depth and breadth and take advantage of multi-disciplinary ties.
- Senior project – The students finish the course with a development or research project.

This format provides flexibility and offers students multiple options/tracks and makes it easier to adapt the course content to remain relevant as IT changes. The flexibility also allows for links with other disciplines to be created, and in some cases working/linking with other departments to achieve this, to show the impact CS has in other areas/disciplines; coverage of the multi-disciplinary nature of CS is rare in other courses. They hope this broader scope will enable students to see more relevance to CS and how it can be used in many areas of industry. The new curriculum had already proved popular after just one year of availability and helped with the noticed 40% increase in major applications. Student feedback was generally positive but they felt that there was a lack of programming which will be addressed in future. The course has also had positive feedback from industry and other universities.

As previously mentioned, hypothesised problems with perception and design of the CS50 course at Harvard (Malan 2010) led to the conclusion that the course needed improving and modernising. The

improved course has seen significant increases in enrolments and the majority of the increase has been female students. The course previously contained a lot less female students than male students, so this increase is very encouraging for a more balanced class. It has even increased enrolments in subsequent CS courses, one increased 33% and another increased 122%!

### *C. Focusing courses around a current trend*

Some institutions have tried to increase course popularity by focusing them around a current trend. Purewal Jr. (2010) explains how there are signs of CS enrolments increasing but this could be short-lived if it is because of a current trend (e.g. social networking). They explain how CS courses could be based around trends and as new computing trends emerge and others lose appeal (for example social networks are replaced by a new trend) they should be refocused accordingly. They believe this approach can maintain and increase CS enrolments and student diversity. The paper focuses around improving the "Communications Technologies and the Internet" introductory CS (CS0) course at the College of Charleston with a focus on social networks due to their current popularity and use of the latest technologies and concepts. They explain the common objectives of CS0 courses and how they believe an additional objective should be added covering "the current ethical, social and legal implications of the growing ubiquity of and increased reliance on technology". They then explain their course and how it meets these objectives. They reflect on the success of the course and conclude that overall it has been successful. A particular highlight that proves the course's relevance was, as the course was being taught, many articles were being published in related areas. This allowed the course to have up to date content to discuss and as technology frequently changes this was very valuable for making the course relevant and current. Student feedback showed there was significant enthusiasm for the course and its contents.

Similarly Kurkovsky (2007) explains the "Introduction to Internet Programming and Applications" course at Central Connecticut State University which introduces the fundamentals of computer programming focussing around the internet and its impact on society. They hope by basing it around a well-known area (the Internet) it will be relevant and motivating for all and make CS more understandable for non-CS majors. Many CS concepts such as network architecture, algorithms, programming etc. can be made more understandable by relating them to the Internet. They found the course was useful for helping increase understanding of CS and motivation to study it.

### *D. Make programming more accessible*

Programming can be difficult for undergraduates to understand especially for non-CS majors and/or those with limited prior experience. Traditional text-based programming languages like Java and C++ can be very confusing as the syntax used isn't easy to

interpret and almost looks like a foreign language. This means students not only have to understand programming concepts but they need to interpret programming syntax. To make introducing programming easier, many visual programming tools/environments were created such as Scratch<sup>117</sup>, App Inventor<sup>118</sup> and Alice<sup>119</sup>. These allow programs to be created by dragging components into the tool instead of writing specific syntax. These components are programming elements such as loops, variables etc. and only fit together in a semantically correct way. This enables students to see how programming concepts such as loops work, without needing to worry about specific syntax and can easily see their mistakes; for example if a component doesn't fit in the chosen location it will alert the user. These tools are intuitive, make programming fun/motivating and are used in CS courses to increase interest in programming, CS courses and careers and to improve course retention and success rates.

Wolber (2011) discusses some initial tutorial examples for Java, Scratch and App Inventor. As Java is text-based and object-oriented it means even the most basic example (displaying "Hello World!" on the screen) involves introducing many complex terms/concepts which are hard to explain to new programming students; they probably won't understand it fully until much later in the course. The initial tutorial examples for Scratch and App Inventor are a lot easier to understand due to the drag-and-drop system. Scratch and App Inventor are very similar and both use blocks (components) that fit together to create the required functionality (e.g. looping through code) and have puzzle style connections that only allow blocks to fit together in a semantically correct way. The main difference between Scratch and App Inventor is that Scratch is contained within the programming tool/environment (although applications can be shared on the Scratch website) whereas App Inventor creates Android applications and can be run on Android mobile devices as well as within its emulator. Due to these reasons as well as for its ability to perform mobile tasks like sending text messages to give applications a real-world purpose, Wolber chose App Inventor for their introductory CS course. It helped the students easily understand programming concepts, quickly create applications with real-world uses and motivated them to tackle more complex programming problems.

Morelli et al. (2010) explains a project to investigate whether App Inventor could be used to teach K-12 students Computational Thinking. It focussed on ideas and lesson plans around App Inventor and created applications that should appeal to the K-12 demographic. The project started with students using App Inventor and then teaching it to some teachers. They conclude that while it is too early to make strong conclusions, it has been a success and App Inventor has proved to be accessible and powerful, can provide an Object-Oriented

Programming model, can be used for problem-driven learning, has motivational potential, is relevant and can support learning.

Uludag et al. (2011) explains a CS0 course that uses Scratch, App Inventor and Lego Mindstorms. They explain the value to App Inventor such as its ease of use, the popularity of Android smartphones and its support for the Lego Mindstorm robotics interface. The course includes interesting practical laboratory style lessons which aim to relate to real-world experiences, be inspirational, motivational and "cool". Due to Scratch being slightly more basic than App Inventor while very similar, they use it to introduce programming prior to the use of App Inventor. They use App Inventor to control Lego Mindstorm robots to make the course more engaging and provide more satisfactory feedback as a result of using programming. They hadn't assessed the courses effectiveness at the time.

Alice is another popular visual programming tool/environment for teaching Object-Oriented Programming (OOP). It is based around 3D animations that demonstrate programming concepts using a simple drag-and-drop system. Many institutions (Mullins et al. 2009; Cooper et al. 2010;) use it as a first programming tool to introduce programming before moving onto other more complex text-based programming languages such as Java and find it is ideal due to its use of OOP. However Adams (2010) considers Alice to be quite complex for initial programming lessons and recommends Scratch is used to introduce programming basics before using Alice. Whereas Malan and Leitner (2007) consider Scratch alone as a suitable basis prior to learning Java. In a similar way to the work by Uludag et al. (2011) Alice can also be used to control robots to make a CS course more engaging (Wellman et al. 2009). Alice has also proven to be useful for transitioning into programming with C++ (Johnsgard and McDonald 2008).

Lewis (2010) evaluates the opinions and learning outcomes of students learning programming using a text-based language (Logo), versus a visual programming tool/language (Scratch). They predicted that because Scratch is visual that students would have a more positive attitude towards it, and consequently programming in general, and have a greater understanding of loops and conditional statements. However they found that Scratch only provided a greater understanding of conditional statements. Also Logo provided students with greater confidence in programming versus Scratch which is opposite to their hypothesis. Students gave both Logo and Scratch a similar difficulty rating and they are similarly motivated to continue programming after using either of them.

#### *E. Different teaching approaches and learning techniques*

Many different approaches and learning styles have been tried to improve student engagement, success rates and interest in CS courses. Many courses have found success by relating their content to real-world examples to help provide context and

<sup>117</sup> <http://scratch.mit.edu>

<sup>118</sup> <http://appinventor.mit.edu>

<sup>119</sup> <http://www.alice.org>

understanding of the value of IT. Uludag et al. (2011) discuss how they believe by basing their course around the constructionist learning theory (learning by doing/making) and active learning with the use of creating Lego Mindstorms robots makes programming more engaging as students can see the effects of it over a physical object. Wolber (2011) however, replaced the Mindstorms element of their course with App Inventor, as mobile applications can provide more relevance to students lives than robots do. Harvard's CS50 course (Malan 2010) uses many learning techniques (lectures, seminars, videos, anonymous bulletin boards etc.) to allow for different learning styles and improve self-learning, problem solving, student engagement, confidence etc. McFarland (2004) identifies three main approaches for teaching CS; breadth-first (covers a wide range of topics to provide a broad introduction to CS), depth-first (focuses on topics in more depth such as a programming focused course) and a blended/balanced approach. Their research led Western New Mexico University to use a balanced approach by starting with breadth-first topics to properly introduce CS and then take a depth-first approach to teach programming concepts. Goldman (2004) introduces a concepts-first approach where their introductory CS course uses JPie (a visual programming tool/environment for creating Java applications) to introduce key CS concepts and software development. Anewalt (2008) uses a non-traditional approach for a CS0 course by using kinaesthetic learning activities including the use of physical props, hands-on labs, competitions and games. The activities (including unusual activities like using playdough to teach classes and objects) are used to help students understand key CS concepts.

#### *F. Outreach projects*

Colleges and Universities promote CS and consequently their courses via various outreach projects; these are typically via introduction/taster courses for high school/secondary school students or by helping their teachers introduce or improve CS teaching.

Adams (2010) explains a summer school outreach program for introducing programming concepts to middle school students. This has been run over multiple years and has proved to be popular and increases awareness of CS, and many students wish to continue learning programming and consider further CS courses and careers.

Cooper et al. (2010) explains a partnership between colleges/universities and middle and high schools as professional development to improve the quality of CS teaching. The pilot project resulted in an improved CS curriculum which was seen as a success and has improved CS lessons and increased CS enrolments.

Egan (2010) explains a one day event/program described as a non-programmer's programming contest designed to show the value of CS to high school students (targeting those with good mathematics and problem solving skills) and their teachers. It focused around group tasks/challenges based around programming skills to provide a fun

introduction to programming. It received very positive feedback from students and teachers and it showed the event had improved perceptions of CS.

Morreale et al. (2010) describes a one day workshop run by a university to help high school teachers teach CS. It was aimed at enhancing CS teaching and improving college/university CS success rates as well as making CS more appealing to students. They also hope that teachers will recommend CS as a further study option and career and ideally recommend study at their university. The workshop was a success as it met these goals.

## VII. CONCLUSION

Although CS enrolments and interest remains low we can see signs of a more positive future with CS enrolments and degree production beginning to rise. There is a lot of work being done on improving perception and understanding of CS via enhanced education, outreach projects, new visual tools for learning programming, online learning etc. Students are more engaged and motivated by these new approaches and there has been improved retention and grades as students see the value and relevance of CS. However it is vital to keep the content modern and relevant to reflect changes in the computing field, including following the latest trends. If a course is based on a popular trend to engage interest and that trend loses popularity in favour of something new, then the course should refocus to cover the new area of interest. The computing environment is constantly changing with regular new innovations which can be a huge attraction for students pursuing CS education. Therefore, course content should adapt to cover the latest computing concepts, technology, trends etc. to remain relevant and retain students' interest.

As governments, industry, professional bodies and educational institutions are realising the need to refocus computing education to being CS focused as well as incorporating ICT, then educators will need to adjust course content accordingly. This is currently very relevant in the UK school system as the new computing curriculum is being introduced replacing the existing ICT curriculum. As previous computing teaching was ICT focused (this typically covered usage of applications like the Microsoft Office suite) teachers may only have learned ICT skills and have no or little CS skills. Teachers will probably need support as they design lessons based on the new computing curriculum and therefore there is a lot of current research around looking at ways to support this process. This could be for example designing course content, finding appropriate tools for teaching specific subjects like programming, assessment methods and so forth.

## ACKNOWLEDGMENT

The author would like to thank Dr Philip Davies for his guidance on writing literature review papers and Dr Sheridan Jeary for her advice on finding a suitable research topic.



## REFERENCES

- ACM/IEEE CS Joint Task Force on Computing Curricula, 2001. *ACM/IEEE Computing Curricula 2001, Computer Science (CC2001)*. New York: ACM. Available from: [http://www.acm.org/education/curric\\_vols/cc2001.pdf](http://www.acm.org/education/curric_vols/cc2001.pdf) [Accessed 09 May 2013].
- Adams, J.C., 2010. Scratching Middle Schoolers' Creative Itch. In: *Proceedings of the 41st ACM technical symposium on Computer science education*, SIGCSE '10, 10-13 March 2010 Milwaukee. New York: ACM, 356-360. Available from: <http://dl.acm.org/citation.cfm?id=1734385> [Accessed 17 May 2013].
- Aspray, W., Mayadas, F., and Vardi, M. Y., 2006. *Globalization and Offshoring of Software: A Report of the ACM Job Migration Task Force*. New York: ACM. Available from: <http://www.acm.org/globalizationreport/> [Accessed 03 May 2013].
- Anewalt, K., 2008. Making CS0 fun: an active learning approach using toys, games and Alice. *Journal of Computing Sciences in Colleges*, 23 (3), 98-105. Available from: <http://dl.acm.org/citation.cfm?id=1295133> [Accessed 17 May 2013].
- Carter, L. 2006. Why Students with an Apparent Aptitude for Computer Science Don't Choose to Major in Computer Science. In: *Proceedings of the 37th SIGCSE technical symposium on Computer Science education*, SIGCSE '06, 1-5 March 2006, Houston, Texas, New York: ACM, 27-31. Available from: <http://dl.acm.org/citation.cfm?id=1121352> [Accessed 17 May 2013].
- Cooper, S., Dann, W. and Harrison, J., 2010. A K-12 College Partnership. In: *Proceedings of the 41st ACM technical symposium on Computer Science education*, SIGCSE '10, 10-13 March 2010 Milwaukee. New York: ACM, 320 – 324. Available from: <http://dl.acm.org/citation.cfm?id=1734371> [Accessed 17 May 2013].
- Department for Education, 2013. *Computing: Programmes of study for Key Stages 1-4*. Department for Education. Available from: [http://computingatschool.org.uk/data/uploads/computing-04-02-13\\_001.pdf](http://computingatschool.org.uk/data/uploads/computing-04-02-13_001.pdf) [Accessed 21 February 2013].
- Dougherty, J.P., Kock, N.F., Sandas, C., and Aiken, R.M. (2002) Teaching the Use of Complex IT in Specific Domains: Developing, Assessing and Refining a Curriculum Development Framework. *Education and Information Technologies*, 7 (2), 137-154. Available from: <http://link.springer.com/article/10.1023%2FA%3A1020305827078> [Accessed 17 May 2013].
- Dougherty, J.P., 2003. Information technology fluency at a liberal arts college: experience with implementation and assessment. *Journal of Computing Sciences in Colleges*, 18 (3), 166-174. Available from: <http://dl.acm.org/citation.cfm?id=771734> [Accessed 17 May 2013].
- Egan, M.A.L., 2010. Recruitment of CS majors through a non-programmer's programming contest. *Journal of Computing Sciences in Colleges*, 25 (6), 198-204. Available from: <http://dl.acm.org/citation.cfm?id=1791165> [Accessed 17 May 2013].
- Forte, A. and Guzdial, M., 2005. Motivation and Nonmajors in Computer Science: Identifying Discrete Audiences for Introductory Courses. *IEEE Transactions on Education*, 48 (2), 248 – 253. Available from: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=1427874> [Accessed 17 May 2013].
- Goldman, K.J., 2004. A Concepts-First Introduction to Computer Science. In: *Proceedings of the 35th SIGCSE technical symposium on Computer science education*, SIGCSE '04, 3-7 March 2004 Norfolk, Virginia. New York: ACM, 432-436. Available from: <http://dl.acm.org/citation.cfm?id=971446> [Accessed 17 May 2013].
- Johnsgard, K. and McDonald, J., 2008. Using Alice in Overview Courses to Improve Success Rates in Programming I. In: *IEEE 21st Conference on Software Engineering Education and Training*, CSEET '08, 14-17 April 2008 Charleston, SC, New York: IEEE, 129-136. Available from: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reload=true&arnumber=4556958> [Accessed 17 May 2013].
- Kurkovsky, S., 2007. Making computing attractive for non-majors: a course design. *Journal of Computing Sciences in Colleges*, 22 (3), 90-97. Available from: <http://dl.acm.org/citation.cfm?id=1181873> [Accessed 17 May 2013].
- Lewis, C.M., 2010. How Programming Environment Shapes Perception, Learning and Goals: Logo vs. Scratch. In: *Proceedings of the 41st ACM technical symposium on Computer science education*, SIGCSE '10, 10-13 March 2010 Milwaukee. New York: ACM, 346 – 350. Available from: <http://dl.acm.org/citation.cfm?id=1734383> [Accessed 17 May 2013].
- Lin, H., 2000. Fluency with information technology. *Government Information Quarterly*, 17 (1), 69-76. Available from: <http://www.sciencedirect.com/science/article/pii/S0740624X99000246> [Accessed 17 May 2013].
- Malan, D.J. and Leitner, H. H., 2007. Scratch for budding computer scientists. In: *Proceedings of the 38th SIGCSE technical symposium on Computer science education*, SIGCSE '07, 7-10 March 2007 Covington, Kentucky. New York: ACM, 223-227. Available from: <http://dl.acm.org/citation.cfm?id=1227388> [Accessed 17 May 2013].
- Malan, D. J., 2010. Reinventing CS50. In: *Proceedings of the 41st ACM technical symposium on Computer science education*, SIGCSE '10, 10-13 March 2010 Milwaukee. New York: ACM, 152–156. Available from: <http://dl.acm.org/citation.cfm?id=1734316> [Accessed 17 May 2013].
- McFarland, R.D., 2004. Development of a CS0 course at Western New Mexico University. *Journal of Computing Sciences in Colleges*, 20 (1), 308-313. Available from: <http://dl.acm.org/citation.cfm?id=1040271> [Accessed 17 May 2013].
- Morreale, P., Joiner, D. and Chang, G., 2010. Connecting undergraduate programs to high school students: teacher workshops on computational thinking and computer science. *Journal of Computing Sciences in Colleges*, 25 (6), 191-197. Available from: <http://dl.acm.org/citation.cfm?id=1791164> [Accessed 17 May 2013].
- Morelli, R., de Lanerolle, T., Lake, P., Limardo, N. Tamotsu, E., and Uche, C., 2010. *Can Android App Inventor Bring Computational Thinking to K-12?* The Humanitarian FOSS Project. Available from: [http://hfoss.org/uploads/docs/appinventor\\_manuscript.pdf](http://hfoss.org/uploads/docs/appinventor_manuscript.pdf) [Accessed 21 February 2013].
- Mullins, P., Whitfield, D. and Conlon, M., 2009. Using Alice 2.0 as a first language. *Journal of Computing Sciences in Colleges*, 24 (3), 136-143. Available from: <http://dl.acm.org/citation.cfm?id=1409900> [Accessed 17 May 2013].
- Naace, ITTE, and the Computing at School Working Group, 2012. *ICT and Computer Science in UK schools*. Computing at school. Available from: <http://www.computingatschool.org.uk/data/uploads/ICT%20and%20CS%20joint%20statement.pdf> [Accessed 21 February 2013].
- Purewal Jr., T.S., 2010. Social Networking: The New Computer Fluency? In: *Proceedings of the 41st ACM technical symposium on Computer science education*, SIGCSE '10, 10-13 March 2010 Milwaukee. New York: ACM, 112-116. Available from: <http://dl.acm.org/citation.cfm?id=1734301> [Accessed 17 May 2013].
- Sahami, M., 2007. *Welcome to the Google Education Summit*. Mountain View: Google. Available from: <http://research.google.com/university/relations/eduSummit2007/Me%20Sahami.pdf> [Accessed 03 May 2013].
- Sahami, M., Aiken, A. and Zelenski, J., 2010. Expanding the Frontiers of Computer Science: Designing a Curriculum to Reflect a Diverse Field. In: *Proceedings of the 41st ACM technical symposium on Computer science education*, SIGCSE '10, 10-13 March 2010 Milwaukee. New York: ACM, 47-51. Available from: <http://dl.acm.org/citation.cfm?id=1734279> [Accessed 17 May 2013].
- Scott Hilberg, J. and Meiselwitz, G., 2008. Undergraduate Fluency with Information and Communication Technology: Perceptions and Reality. In: *Proceedings of the 9th ACM SIGITE conference on Information technology education*, SIGITE '08, 16-18 October 2009



- Cincinnati, Ohio, New York: ACM, 5-10. Available from: <http://dl.acm.org/citation.cfm?id=1414562> [Accessed 17 May 2013].
- Uludag, S., Karakus, M. and Turner, S.W., 2011. Implementing IT0/CS0 with scratch, app inventor for android, and lego mindstorms. In: *Proceedings of the 2011 conference on Information technology education*, Sigite 11, 20 - 22 October 2011 New York. New York: ACM, 183-190. Available from: <http://dl.acm.org/citation.cfm?id=2047645> [Accessed 17 May 2013].
- United States Department of Labor, Bureau of Labor Statistics, 2007. *Employment Projections: 2006-16*. Washington: United States Department of Labor, Bureau of Labor Statistics. Available from: [http://www.bls.gov/news.release/archives/ecopro\\_12042007.pdf](http://www.bls.gov/news.release/archives/ecopro_12042007.pdf) [Accessed 03 May 2013].
- Wellman, B. L., Davies, J. and Anderson M, 2009. Alice and Robotics in Introductory CS Course. In: *Proceedings of The Fifth Richard Tapia Celebration of Diversity in Computing Conference: Intellect, Initiatives, Insight, and Innovations*, Tapia '09, 1-4 April 2009, Portland, Oregon. New York: ACM, 98-102. Available from: <http://dl.acm.org/citation.cfm?id=1565822> [Accessed 17 May 2013].
- Wolber, D., 2011. App inventor and real-world motivation. In: *Proceedings of the 42nd ACM technical symposium on Computer science education, SIGCSE '11*, 9-12 March 2011 Dallas. New York: ACM, 601 - 606. Available from: <http://dl.acm.org/citation.cfm?id=1953329> [Accessed 17 May 2013].
- Zweben, S., 2009. 2007-2008 Taulbee Survey. *Computing Research News*, 21 (3), 8-23. Available from: <http://cra.org/uploads/documents/resources/crmdocs/issues/0905.pdf> [Accessed 17 May 2013].
- Zweben, S., 2010. 2008-2009 Taulbee Survey. *Computing Research News*, 22 (3), 7-24. Available from: <http://cra.org/uploads/documents/resources/crmdocs/issues/0510.pdf> [Accessed 17 May 2013].
- Zweben, S., 2013. 2012 Taulbee Survey. *Computing Research News*, 25 (5), 11-60. Available from: <http://cra.org/uploads/documents/resources/crmdocs/issues/0513.pdf> [Accessed 17 May 2013].

## Appendix 2 – Communications with local schools

### Initial email

Below is a sample of the content of emails sent to schools to initiate conversations regarding computing at their school (the contents were personalised slightly for each school).

“I am currently working on my dissertation project as part of a master’s degree in Enterprise Information Systems at Bournemouth University and wonder if you can help me. The project is based around assisting teachers with the teaching of computing in secondary schools which I assume will be especially useful at the moment given the proposed changes to the computing curriculum to focus on computing rather than just ICT. I have been researching various options for improving and modernising the teaching of computing to enhance the perception and understanding of computing to show its value, relevance and appeal. The next stage will be to assess areas teachers would appreciate some assistance with when teaching computing. Then from this feedback I will research solutions to any problems and create artefacts that will benefit schools such as course content, lesson plans, guidelines, applications and so forth. Therefore I would appreciate it if I could talk to you and perhaps some of your colleagues about how I could assist [school name] and other secondary schools with the teaching of computing.”

### Replies received and subsequent conversations

Below are contents of replies received from the initial emails and subsequent conversations.

Note: Conversation details are not word for word transcripts and email content is not entire emails to leave only relevant content (for example details like arranging meetings are removed).

#### *St Edward’s School*

St Edward’s School (<http://www.st-edwards.poole.sch.uk>) is a joint Roman Catholic - Church of England voluntary aided secondary school in Poole, Dorset. Their subject leader for Computing and ICT, Alastair Barker, replied to the initial email with the following information.

“I have to say that computing has been my focus since arriving in 2007 and has been steadily embedded into the curriculum until now we only teach ICT to a single option group at GCSE. Computing is taught throughout years 8 and 9 (and soon to be 7) and we offer GCSE Computing as well as an established A-level. You can read about us in the latest CAS SwitchedOn magazine (page 4) (Barker 2013).

It would be very interesting to talk to you about your thoughts and research. We would certainly like to access to any experiences you at the University could provide. It is difficult giving the students actual context and examples of the real application of computer science. BU has an impressive international reputation in the world of technology and to be able to visit and see what is going on would be the first thing that I would like to request.”

Alastair had previously met with Stuart Wray, Senior Lecturer at the Royal School of Signals in Blandford, to discuss how to make computing more appealing to students and considered the possibility of a computer club to assist with this. The author met with them on their second meeting to continue these discussions. Topics discussed included:

- How do you make students want to learn computing? What is their motivation – why should I learn computing?
  - Because it's useful?
  - Peers, parents etc. having and using computing skills?
    - Does adults having and using computing skills make it cool or have the opposite effect?
  - Job prospects
  - More opportunities
  - Industry demand
- How can we help teachers to teach computing without being patronising?
- How to make computing fun, motivating and appealing.

The main idea that came out of this meeting is to create a schools outreach event run by St Edward's school with assistance from Bournemouth University for local schools' students (probably Key Stage 2 students) and their teachers. The event will be designed to make programming more fun and engaging for students by showing the effects of programming over a physical object for example turning on a light, interacting with sensors etc. and how inputs such as switches can be used. For example by creating a home scenario where programming (probably in Scratch) is used to control lights, heating etc. or perhaps a car scenario controlling traffic lights and perhaps expanding it to consider a pedestrian crossing. Each device would be connected to a controller (Arduino, Raspberry Pi, network etc.) and the students would need to write some code (either from scratch or by modifying or completing existing code) to control the devices.

In a later email discussing his outreach plans Alastair summarises the areas he wishes to work on/improve.

“I therefore have x2 main areas I want to build on next academic year:

- bring context and real world application of computing to inspire and engage all students eg.
  - increasing physical computing into the curriculum (Arduino/RPi/robotics etc.)
  - increasing profile of computing around school
  - guest speakers
  - trips to institutions and companies that use computing
  - increasing links with companies for apprenticeships/employment
- increasing coding skills amongst staff and students primarily in our x22 feeder schools, but also to all schools in the Poole/Bournemouth area
  - various CPD and school based initiatives including setting up a mock home automation (see below)

I am particularly excited about an event that Paul, Stuart and I developed yesterday. The event will be based around a mock-up of a home (I'm thinking along the lines of putting lamps, radio, TV, curtains/blinds around the Computing block but open to suggestions). Each device will be connected to a controller (eg. Arduino/RPi/network) and using Scratch, students can program these devices to do something - turn on/off most likely. The more able students may be able to do more sophisticated things - this event is certainly a work in progress but meets my need for both CPD and engagement.”

The author then asked for clarification about outcomes he expected from the outreach event, how he thought it might work and so forth; Alastair replied with:

“I've looked at a number of options and have chosen, to start with, traffic lights. In groups of x4, students will create their own set of traffic lights using Arduino/RaspberryPi and Scratch as the coding language. Tasks will range from simply creating a working sequence, adding a pedestrian crossing, adding different sequences for different times of the day through to putting all the traffic light systems together to form a small towns road network.

I am currently talking with Siemens to add the icing on the cake as their traffic light division have a schools outreach program and go into schools with a traffic light system for students to program. The grammar school already use this service if you need to get some empirical evidence.

If this is a success then I will work on the home automation idea. The main reason for not going ahead with this idea is that working with electricity is dangerous and needs expensive equipment to make safe and also there is less scope for feedback loops.

My success criteria would be that students are more confident to code and want to learn more, and that staff understand the principles of coding and are more familiar/confident with Scratch coding. So I guess you will have to question both staff and students at the start and end of session to see if they are more/less confident and willing to learn more about coding.”

### *Poole High School*

Poole High School (<http://poolehigh.poole.sch.uk/>) is secondary school in Poole, Dorset.

Kane Lean, one of their computing teachers, replied to the initial email with the following information.

“In September we are to launch the GCSE Computing course (AQA exam board) with our first classes so your email has actually come through at an ideal time. I do feel we have quite a good handle on our Computer Science teaching methods, however as a department we are very open to new ideas and trying new approaches. As a brief overview, we currently deliver computing through Scratch and Small Basic with years 8 & 9, moving towards Visual Basic and Google App Inventor (MIT) with Key Stage 4. At A-Level we stick with Visual Basic but we do teach basic PHP as part of a web development unit.

It does indeed sound like your dissertation has the potential to be very interesting. It would be ideal to get an outline of the steps you intend to take and any resources you may need from us (i.e. meetings with timings etc.)”

The author then replied with more detail on the project/dissertation and related literature review as requested. Kane then replied with:

“I had a read through your literature review and there are a couple of additional issues which you may wish to consider that I've often found to be a problem in CompSci teaching in schools. Primarily they relate to hardware and network support. Some school systems are understandably locked down which can make executable / binary generation an issue. Network filtering and port blocking can also be problematic (particularly in regards to app inventor, but also other online resources). Fortunately we're lucky to have a good IT support team, but I do know that most schools in the UK are not in the same position as us.

A couple of the points you mentioned which I believe we can relate to are the "Making computing cool" and "make learning about how computers work fun". We do occasionally struggle with these two points in particular at Key Stage 3. I should also mention that we have a full compliment of Raspberry Pi's but we've been unable to find a suitable way to deploy them thus far.”

The author then met with Kane and discussed the idea of creating an event to improve opinions and understanding of computing. The event is similar to the event idea discussed with St Edward's but it was felt that given the limited time available before students are away for the summer holidays it would be more feasible to focus the event around year 8 students. The reasoning being secondary schools have dedicated computing staff (this is rare in middle schools) so it is easier to contact staff who would be interested in the event and could arrange that their students attend.

Kane would like a way of helping students appreciate how computers work rather than just how to use them such as understanding the hardware inside a computer, how to program and so forth. In addition it would be ideal if the event helped students see the relevance of computing with real-world applications and examples. They have a set of Raspberry Pi computers and would like to see how they could use them and the event could be based around these.

Goals, tasks and ideas for the event were also discussed such as:

- It must happen before 22<sup>nd</sup> July as this is when the school term ends and they will need a minimum of 2 weeks' notice to allow them to make arrangements for the students to attend.
- Student differentiation needs to be considered - The event should be inclusive and challenge all students with activities at an appropriate level for varied skill levels within the group.
- Teams would help with inclusiveness and differentiation as teams can have a variety of skills and therefore everyone can learn from each other and the difference between skill levels would be less of a problem.
- Mixing teams up between schools would provide more equal skills within the teams and therefore there are no teams with greater knowledge than others (some schools may cover more computing content than others).
- Teams could present their findings, what they learned, something they made and so forth.
- Perhaps the event should have multiple activities to provide variety rather than one long activity that covers the entire event.
- Perhaps have one Raspberry Pi per team
- It should be a basic introduction to computing to allow for those with little or no prior computing skills.
- Perhaps Bournemouth University outreach and advertising/marketing will be willing to help, maybe even sponsoring prizes.
- Tasks
  - Locate a venue - We can use Poole High School as a venue but it is felt that Bournemouth University is a more suitable venue.

- Decide on a year group to take part in the event. Years 8 and 9 from Poole High School are available until 22<sup>nd</sup> July and Year 8 is probably ideal due to their limited computing experience.
- Plan the event.
- Invite schools.
- Work out the ideal number of students per event (12-15 students is a preferred size of group for Poole High School to bring to the event).
- Think about what will be achieved, what are the events aims and how will they be achieved?
- Find people who will help run the event and decide what they will need to do at the event.
- Work out how to measure success and if it is via observations work out what is to be observed and who will do the observations.
- Work out how teachers would be involved? Some may not understand some technology especially modern technology like a Raspberry Pi or an Arduino so content will need to be basic.
- Decide how long it will run for, all day may be too long, perhaps make it 4 hours.
- Decide on how to conclude the event, perhaps have presentations with prizes.
  - If providing prizes who will pay for them?

The author also asked Kane the following questions:

**Q:** How have you interpreted the new National Curriculum? For example are you making major changes or just adjusting your ICT subject's content? I assume this is covered by the new GCSE but are you making any other changes?

**A:** We have implemented a new GCSE computing subject, modified the computing A-Level (mainly to deal with the removal of January exams) and added programming for every year.

**Q:** Do you feel you have enough information about the new National Curriculum and what is expected of you?

**A:** Yes

**Q:** Are you anxious about the changes being made and do you feel confident in your ability to deliver the required content?

- Do you know all the subjects in the new National Curriculum?
- Does the new content worry you?

**A:** No, we are happy with it.

**Q:** Are you ready for this September? Will you be changing content from September 2013 or will you wait until 2014 when it is compulsory?

**A:** Yes, we are phasing in new computing content over 3 years which will be complete by September 2014.

**Q:** Do you feel you will have everything ready for September 2014 or do you need more assistance to meet this deadline?

**A:** Everything will be up and running by September 2014 due to the phased implementation of computing content which started in 2012 so everything will be fully switched over by 2014.

**Q:** Are you aware of the resources available to help you teach the new computing curriculum?

**A:** Yes via CAS, TES etc.

**Q:** Would you appreciate guidance on how to meet the aims of the new National Curriculum and resources available to you?

**A:** Yes as long as it was just guidance; it is always good to read about different opinions, see other approaches etc.

**Q:** What content do you cover?

**A:** All content specified in the GCSE Computing qualification requirements and the A-level Computing qualifications requirements. Details can be found on the AQA website.

**Q:** How popular is the computing GCSE option? What percentage of students chose to study it?

**A:** 50 year 9 students (2 classes, we would like to increase it to 3 but have a recruitment problem) are to go into year 10 GCSE Computing in September out of 330 students (15.15%).



**Q:** I assume GCSE ICT is/was offered, is/was it popular? What percentage of students chose to study it?

**A:** No, we offer a BTEC ICT course which will be retained alongside the GCSE Computing course.

**Q:** How popular is the AS/A2 course? What percentage of students chose to study it?

**A:** 25 (1 class, would like to increase it to 2) out of 140 students (17.86%).

**Q:** How many hours a week do students learn computing? Will it change with the new National Curriculum?

**A:** 5 hours a fortnight for GCSE, 9 per fortnight for A-Level. It won't increase.

**Q:** Do you have an outreach program with the schools that feed into your school i.e. year 6?

**A:** No

**Q:** What level of prior computing knowledge do students have when they join the school? Is it self-taught or via school?

**A:** Very little, they may have used Scratch but in general their knowledge is poor. It is mostly self-taught knowledge; most will come with some skills but of a very low level.

**Q:** If I were to create a resource such as a workshop, course content etc. would it be possible to test it with the students (probably in mid-July or early August) or will they be on holiday and won't be interested in coming into the school then?

**A:** Yes until 22<sup>nd</sup> July

**Q: And most importantly:** Can I have permission to publish our conversations in my dissertation?

**A:** Yes as long as it doesn't include any personally identifiable details/data about the students.

## References

Barker, A., 2013. Introducing Computing: A Success Story from Dorset. *Switched On*, Summer 2013 (1), 4. Computing at School Working Group. Available from: <http://www.computingschool.org.uk/data/uploads/newsletter-summer-2013.pdf> [Accessed 3 June 2013].

### Appendix 3 – Surveying teachers via an informal discussion

The author created an informal discussion on an online forum to obtain details on how prepared teachers are for the computing changes in the National Curriculum and whether more help would be useful.

#### Initial post

The author's initial post contained:

“I am currently working on my dissertation project as part of a master's degree in Enterprise Information Systems at Bournemouth University and wonder if you can help me. The project is based around assisting teachers with teaching computing and for this I need to establish how prepared teachers are for the new computing curriculum and if they would appreciate any help. This will allow me to focus my research onto areas that will assist teachers and hopefully create useful guidelines, resources etc. Therefore can you please tell me more about your views on the new curriculum and areas that you would appreciate some help in. For example:

1. How have you interpreted the new curriculum? For example are you making major changes or just adjusting your ICT subject's content?
2. Do you feel you have enough information about the new curriculum and what is expected of you?
3. Are you anxious about the changes being made and do you feel confident in your ability to deliver the required content?
  - o Do you know all the subjects in the new curriculum?
  - o Does the new content worry you?
4. Are you ready for this September? Will you be changing content from September 2013 or will you wait until 2014 when it is compulsory?
5. Do you feel you will have everything ready for September 2014 or do you need more assistance to meet this deadline?
6. Are you aware of the resources available to help you teach the new computing curriculum?
7. Would you appreciate guidance on how to meet the aims of the new curriculum and resources available to you?

Please provide details of your current and future teaching. Even if you feel you have everything covered please still explain it as it will be beneficial to my research and will probably help other on here too.”

#### Replies received

The following replies were received.

#### *Response from Jules Thompson, Head of ICT, St Thomas More RC School, Buxton, Derbyshire*

“Just to put my views into context. I am Head of ICT at a small school in Derbyshire, I am the only teacher of ICT in the school. I have been teaching ICT for 11 years (HoD for 7). I came to ICT via. English, which is my degree and PGCE subject. ICT has always been a very popular and successful subject at our school; in the top rankings for residuals, progress, uptake etc.

1. I have planned a major shake-up of our KS3 curriculum. I've trimmed all the fat and repetition from the old ICT curriculum and created six 'threads' which will span the three years of KS3. The threads are: \_Communication, Data Handling, Programming, Computer Systems, Computers & Society and E-safety\_. This incorporates the best and

most valuable aspects of the dis-applied ICT curriculum and starts to phase in the new draft Computing curriculum at a pace we can cope with in our own setting.

2. I would like more information on what's expected from the new curriculum, for example my school still insists on me using NC levels (even though they are meaningless at the present moment in time) so I would like to have a hint of level descriptors to see rate of progress etc. I would also like some official confirmation that we're not being expected to deliver the entire draft curriculum to ALL pupils from Sept 2014 and some understanding that it needs to be a more gradual process.

3. I have been extremely anxious about the new curriculum, to the point of seriously considering my alternatives to teaching (and I LOVE my job). Initially it was overwhelming and I used the analogy that I was like a PE teacher suddenly being asked to teach Mandarin to GCSE level, with no training or support. Glad to say I've gained a bit of perspective since then, but I still think it's a massive shift and one that's not been well thought out. I'm not very confident in my ability to deliver all of the content immediately; but I am now at least a bit more confident that I have a strategy in place to address this (thanks in no small part to advice I've had on CAS online).

4. I will be introducing lots of new content from September 13; HTML in Year 8, Python in Year 9 along with lots of hardware, software, networks, and emerging technologies across all three year groups. I'm not worried about the new content as such, it's more an issue of time to get myself up to speed and get schemes of work in place etc. And for those who think I can't do it all I'd say is that we didn't do much in the way of relational databases in my English degree but I've been confidently and successfully teaching them for the last decade. Fortunately I have quite a supportive head who is giving me one lesson a week for training next year.

5. We won't be covering everything by September 14, it's just not realistic. I have no one to delegate anything to, I need to retrain myself and it simply can't be done in that time scale. I also have a work/life balance to consider (not to mention a toddler at home.) More assistance would always be welcomed, but at least I'm now confident in the position we have taken and feel I could justify it if necessary.

6. Accessing resources has been my biggest concern, I hate to say that the LA have been very poor on this score. Fortunately I have sourced loads of stuff from CAS and they have pointed me in the direction of Udacity to help me gain some knowledge and confidence in Computer Science.

7. Any guidance always gratefully received.

I would also be happy to email you the new KS3 scheme I've put together, although it's still a work in progress."

The author then asked some additional questions and responses are below:

**"1. Do you have or had in the past an ICT GCSE option?**

Yes, we offered GCSE ICT as an option up until 2009 when we switched to OCR Nationals. As this is only a small school, we only offer the one option course.

**If so:**

**Is/was it popular?**

It was very popular.

**What percentage of students chose to study it?**

On average around 70% of each cohort and a pretty even balance of males and females too. (The uptake for Nationals remained around the same, apart from a dip in the current Year 10, which has become known as the 'Ebac effect', the option groups for that year are completely different from the usual pattern with a massive increase in the uptake of languages and humanities and a fall in ICT, catering, Graphic Products and the art subjects. Now Ebac has all but disappeared we've reverted back to the usual distribution.)

**Is it still running?**

We have reverted back to GCSE ICT for this September as I was very concerned about the longevity and credibility of the Cambridge Nationals.

**2. Do you have or plan to have a computing GCSE option?**

In the short-term no, although hopefully in the next 3-5 years as we become better equipped (mostly through CPD) to deliver it.

**If so:**

**How popular is the computing GCSE?**

**What percentage of students chose to study it?**

**Will it replace the ICT GCSE or remain as an additional option?**

**3. Do you have a computing AS/A2 course?**

No, we are 11-16 only

**If so:**

**How popular is the AS/A2 course?**

**What percentage of students chose to study it?**

**4. How many hours a week do students learn computing? Will it change with the new curriculum?**

They have one lesson of 50 minutes each week at key stage 3 (years 7-9)

If they choose ICT as an option they have 3 x 50 mins per week

Plus ALL pupils in Year 10 & 11 do some ICT as part of a 'carousel', approx. 20 lessons in Year 10 and 8 in Year 11.

I am not aware of any planned changes to the allocation for ICT/Computing in the timetable, certainly there is no change from above for 2013\_2014.

**5. Do you have an outreach program with the schools that feed into your school i.e. year 6?**

We are supposed to, but it has kind of faltered over the last few years. Every year around this time I think, I really must do something about this, but it's difficult with all the changes going on currently and it's not high on the priority list.

**6. What level of prior computing knowledge do students have when they join the school? Is it self-taught or via school?**

Depends which feeder primary they come from! Most have basic transferable skills in MS Office, mostly focusing on formatting and working with images. Some have used Flowol at primary so have a basic understanding of control, but that's about it. I think it's as much home as it is school, but again it depends on the background of the individual child; some are arriving having already created and published their own websites, but these are in the minority!

**7. And most importantly: Can I have permission to publish our conversations in my dissertation?**

Yes that's fine."

## Appendix 4 – Acquiring knowledge to teach the new National Curriculum

As discovered in the investigation into computing in schools (see appendices 2 and 3) some computing teachers may not have computing qualifications and are concerned about teaching the new content<sup>120</sup>. There may also not be enough awareness in schools of the changes and colleagues may not understand the scale of the transition to the new Computing subject. Equally there may not be support by executives<sup>121</sup> to assist computing teachers with training/re-training such as funding courses or providing time to learn. In both situations this can be improved by greater communication to explain the changes to computing education in schools; also awareness in general is improving as more details emerge on the new NC<sup>122</sup>. There is also a lot of support for training new computing teachers such as generous scholarships such as the BCS scholarship (BCS 2013).

### References

BCS, 2013. *Teaching scholarships / BCS Academy of Computing*. Swindon: BCS. Available from: <http://academy.bcs.org/scholarships> [Accessed 12 August 2013].

---

<sup>120</sup> For example see the comments from Jules Thompson in appendix 3 who was so concerned and anxious about the extra content to learn that she considered changing careers.

<sup>121</sup> Such as head teachers, heads of departments and year groups, governors, managers, local and national government and so forth.

<sup>122</sup> The new NC covers changes for multiple subjects so it is well known within schools but the exact changes to subjects such as the change from ICT to Computing may not be as well understood.

## Appendix 5 – The new National Curriculum

The new Computing subject in the new National Curriculum was influenced by reports/proposals/arguments such as:

- The ‘Next Gen.’ report (Livingstone and Hope 2011)
- The ‘Shut down or restart? The way forward for computing in UK schools’ report by The Royal Society (2012)
- The ‘ICT and Computer Science in UK schools’ report by the Naace, ITTE, and the Computing at School Working Group (2012)
- A proposal to include CS in the English Baccalaureate by an expert panel led by BCS, The Chartered Institute for IT (BCS 2012)

BCS (2013) responded to the new NC proposals regarding the Computing subject on behalf of BCS and CAS and their respective members. Overall they supported the changes and welcomed the renaming of ICT to Computing to emphasise the change in direction and improvements. However they did make some further suggestions such as including teaching eSafety citing the concerns raised by CAS members.

The DfE (2013) have made GCSE Computer Science a separate science option (bringing the total science options to 4) and included it in the English Baccalaureate (EBacc) performance measures<sup>123</sup>.

### References

- BCS, 2012. *The case for Computer Science as an option in the English Baccalaureate*. Swindon: BCS. Available from: <http://academy.bcs.org/sites/academy.bcs.org/files/Case%20for%20Computer%20Science%20as%20an%20EBacc%20option.pdf> [Accessed 6 June 2013].
- BCS, 2013. *BCS, The Chartered Institute for IT in association with the Computing At School group Consultation Response to: Reform of the National Curriculum in England*. Swindon: BCS. Available from: <http://academy.bcs.org/sites/academy.bcs.org/files/BCS%20National%20Curriculum%20Response%20-%20April%202013.pdf> [Accessed 4 July 2013].
- DfE, 2013. *Computer science to be included in the EBacc*. Department for Education. Available from: <https://www.gov.uk/government/news/computer-science-to-be-included-in-the-ebacc> [Accessed 07 June 2013].
- 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*. Next Gen Skills. Available from: <http://www.nesta.org.uk/library/documents/NextGenV32.pdf> [Accessed 6 June 2013].
- The Royal Society, 2012. *Shut down or restart? The way forward for computing in UK schools*. London: The Royal Society. Available from: <http://royalsociety.org/education/policy/computing-in-schools/report/> [Accessed 6 June 2013].
- Naace, ITTE, and the Computing at School Working Group, 2012. *ICT and Computer Science in UK schools*. Computing At School. Available from: <http://www.computingsatschool.org.uk/data/uploads/ICT%20and%20CS%20joint%20statement.pdf> [Accessed 6 June 2013].

---

<sup>123</sup> These GCSEs are approved by BCS, the Chartered Institute for IT and the Royal Academy of Engineering (RAEng) to ensure they are of the required high quality (DfE 2013).

## Appendix 6 – Programming languages/tools/environments for education

### Visual programming languages/tools/environments

#### *App Inventor*

App Inventor allows easy creation of applications for Android devices which provides huge motivational potential and ability to create real-world and relevant applications. It has a superb website which includes in-depth and well written documentation and tutorials to help understand the tool and easily create applications.

The first tutorial “HelloPurr” shows how to add a button with an image on it and add a sound which is played when the button is pressed. It provides a full introduction to how App Inventor works and creating an application as well as the basics of application development. Further basic tutorials continue from this by introducing other programming concepts such as variables, random choices, loops etc. There is an area for teachers containing lessons which advance in difficulty and create a small course. The first lesson<sup>124</sup> “Magic 8 ball” introduces random choices after a button click<sup>125</sup> to output a random prediction just like shaking a magic 8 ball. It also introduces the use of the accelerometer sensor to allow the user to shake the phone to get a prediction which replaces the button click. The author also found it easy to create a function<sup>126</sup> containing the prediction code to allow it to be used with a button click and the accelerometer sensor. They suggest possible additions to try such as responding to text messages with a prediction. The author found it is simple to work out how to do this and it allows for the introduction of ‘if statements’ so that it only responds when the text message sent to it starts with the keyword ‘magic’. It can also be used to introduce text formatting functions<sup>127</sup> and comparison functions<sup>128</sup>. Adding a button to close the application was also simple. The author also found it was easy to create another example of displaying times tables of a number chosen from a drop down list which introduces the use of ‘for’ loops.

---

<sup>124</sup> However it would be sensible to start with the “HelloPurr” tutorial prior to this to help students understand how App Inventor works.

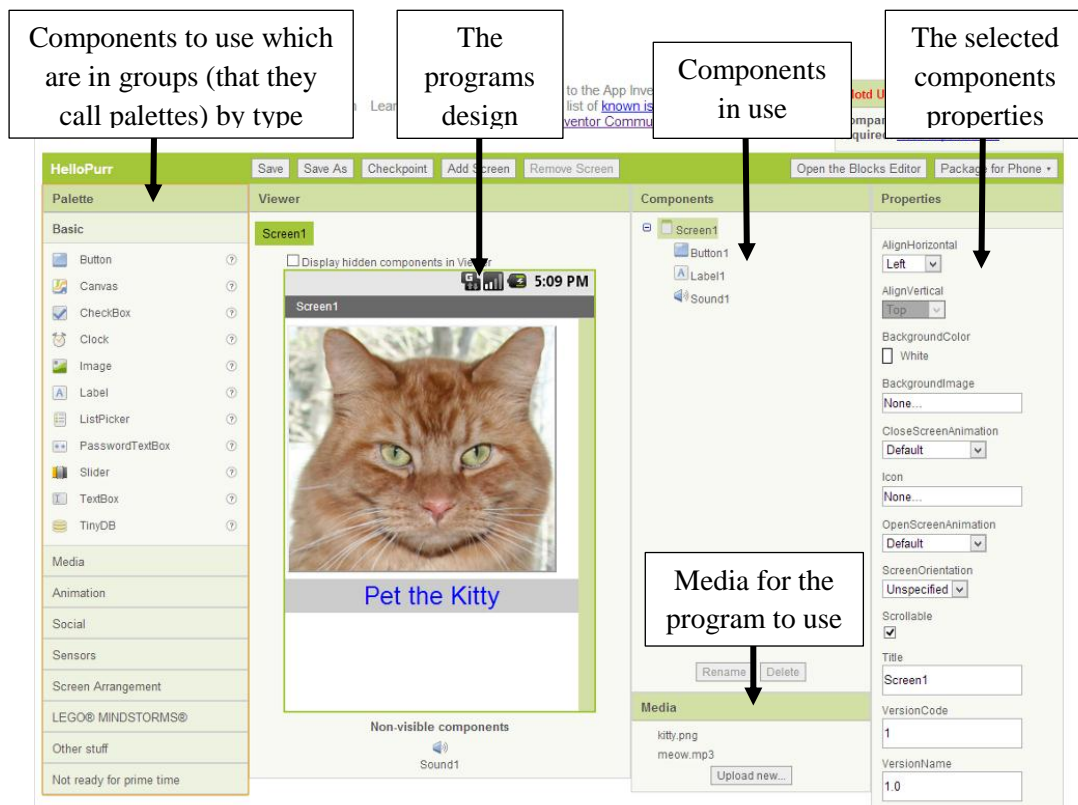
<sup>125</sup> Which has a picture of an 8 ball on it

<sup>126</sup> App Inventor calls functions procedures

<sup>127</sup> For example to remove the need for case sensitive text messages when identifying the keyword

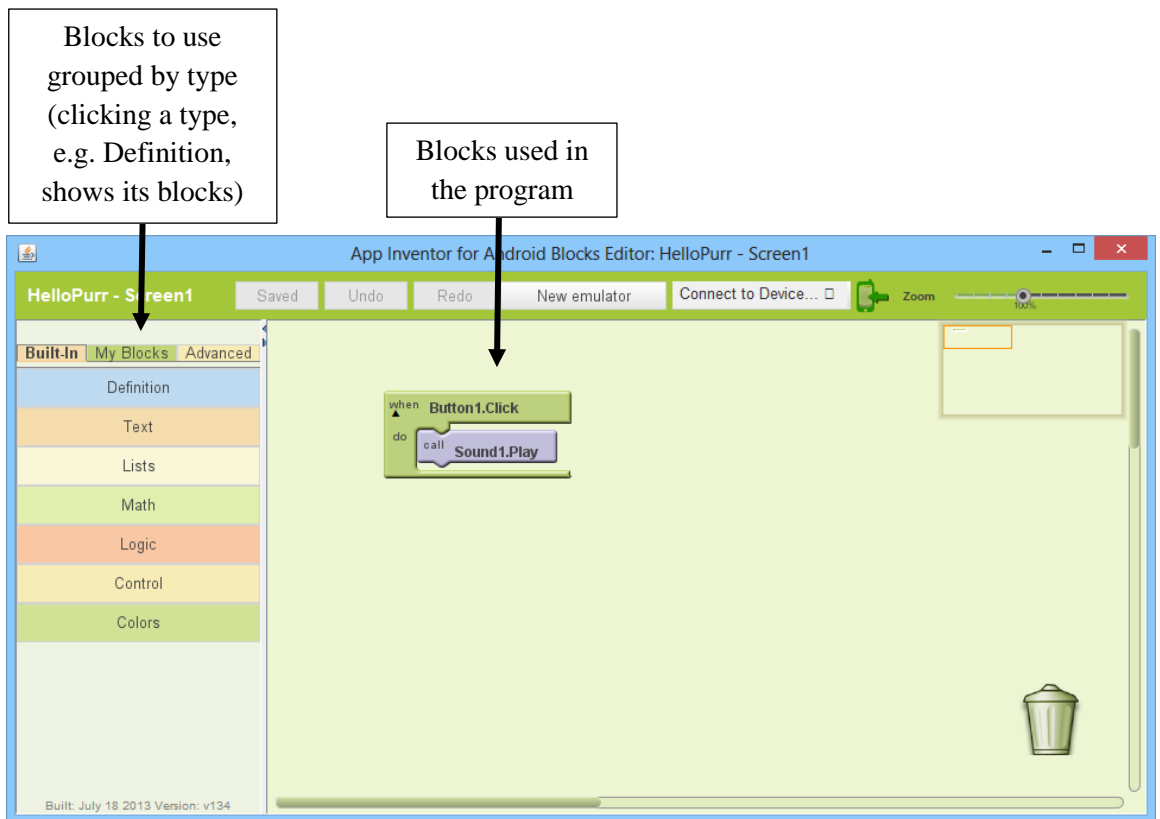
<sup>128</sup> For example to check for the keyword in the text message received

It is easy to create and store programs in App Inventor due to it being web-based (although it does require installing the App Inventor program and also the blocks editor requires the downloading and use of a Java web start file) with applications stored online for easy access. The use of puzzle piece blocks/components makes identifying suitable connecting components easy. However the author found that some components which shapes/connections appear to suggest fit together prompted an error; for example a component that only allows numerical inputs will show an error if a text input is added which while it teaches the student about data types it could be confusing and frustrating. Another useful feature is live development/testing which allows an Android device that is connected to the computer or is on the same network to be used to test the program being worked on in App Inventor. Changes made in the code during development immediately affect the development version of the program on the device so it is ready to be tested; this provides immediate feedback and encourages and simplifies testing. Figure 1 shows the interface designer and figure 2 shows the blocks editor.



**Figure 1: App Inventor - Interface Designer Screen**





**Figure 2: App Inventor – Blocks Editor**

**Usability of the programming tool/environment: 3.5**

The drag-and-drop interface for both the screen designer and the blocks editor combined with the puzzle piece connections makes designing applications and adding functionality straightforward. The layout is clear and its ability to set positioning and attributes of elements without needing to use coding simplifies designing interfaces. The ability to connect an Android device for live development/testing during development adds more realism and simplifies the testing process. The requirement to download a file to open a program in blocks editor slightly increases complexity and more help/tips within the environment would be useful.

**Ease of use and intuitiveness of the programming language: 4**

The use of puzzle piece blocks vastly simplifies programming by removing the need to write code. However more detail on choosing the right component would be useful along with a bit more detail on how they work.

**Programming concepts covered: 3.5**

Most concepts are included in a simple way; however advanced concepts like OOP are not covered.

**Relevance to industry: 3.5**

While it probably isn't used extensively in industry due to its simplicity it may be used for prototyping or creating basic Android applications. Also creation of Android applications is common in industry.

**Ability to create real-world and relevant applications: 4**

Despite its simplicity it is able to create real-world and relevant Android applications; however it doesn't cover every Android feature.

**Interactive features: 4.5**

There are components for using most features of Android devices such as texting, speech recognition, barcode scanners, sensors and so forth. It also has the ability to connect to Lego Mindstorms kits to control physical electronic components such as sensors, robots and so forth.

**Motivational potential: 4.5**

Its ease of use, ability to create real-world and relevant applications, use of Android devices and their functions and other interactive features make it very motivational and appealing.

**Quality of documentation and amount of teaching resources available: 4**

There are many excellent and easy to understand tutorials and resources available but more help/tips within the environment would be useful.

**Longevity and update frequency: 4.5**

It is developed and maintained by the Lifelong Kindergarten Group (2013) at MIT's Media Lab (MIT Media Lab 2013a) and is regularly updated and has support to keep the project running.

**Total: 36 (80%) Average: 4 Weighted Total: 107 (79.26%) Weighted Average: 3.96**

### *Scratch*

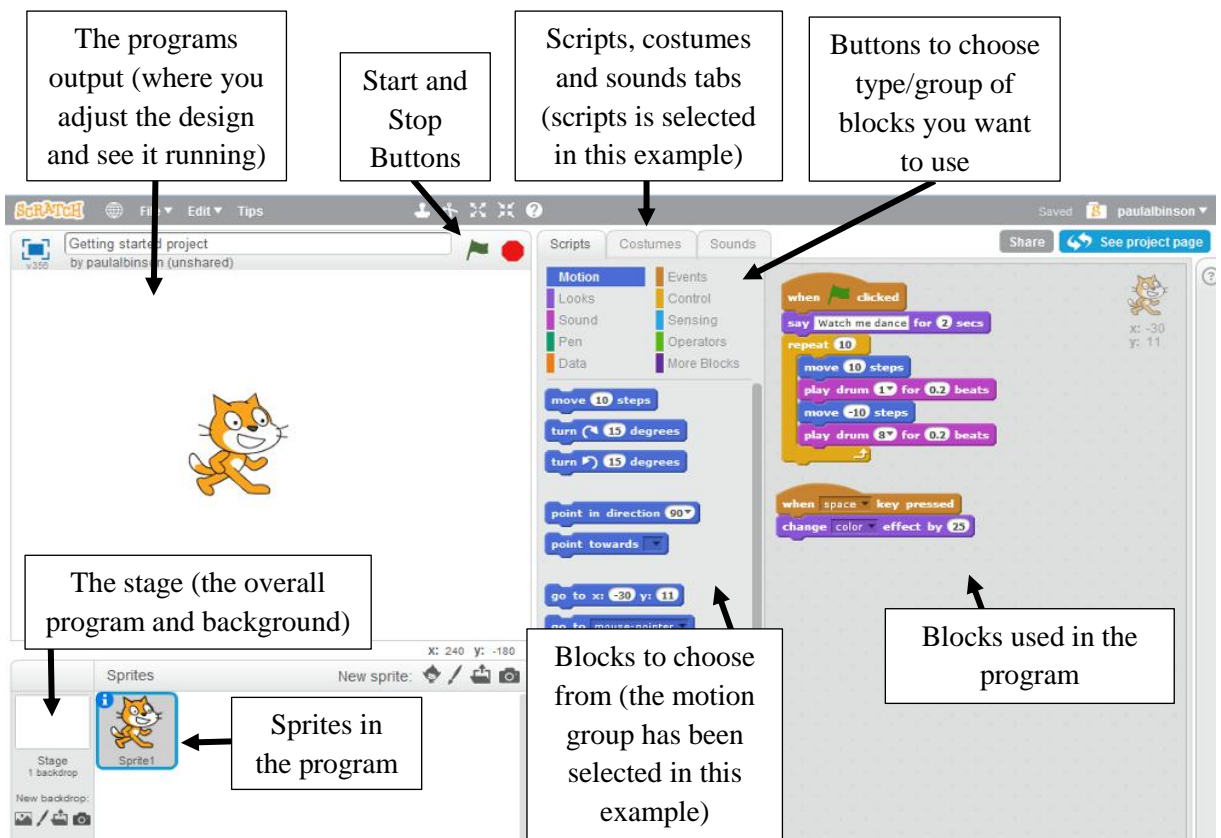
Scratch works in a similar way to App Inventor and shares the same puzzle piece style blocks system. Scratch is primarily designed for a younger age range (8-16 year olds) but can be used to teach programming to almost any age perhaps as an introduction to programming concepts prior to using other languages and tools such as Alice, Java etc. Sharing projects and ideas is done via the Scratch website rather than mobile devices. Scratch 2.0 has recently been released with many improvements such as the ability to define blocks (creating your own functions) and making it cloud-based to allow programs to be created and stored online<sup>129</sup>. Users can also look at the code of shared projects to learn how they work and to modify/remix them to create their own version of the project. It also allows for more interactivity with the ability to track movement via a webcam and they plan to add integration with external devices including ScratchBoard<sup>130</sup> and Lego WeDo. An interactive getting started tutorial is available to introduce how Scratch works and create a basic project. It is easy to use and makes Scratch easy to understand, however some steps aren't mentioned such as which category the required block is in but it does show it in an image and the blocks are colour coded. There are lots of other tips and information on how to use Scratch in a tips tab in the editor so it is easy to find help while creating a project<sup>131</sup>. There is a vast amount of information, sample projects and resources available for teaching including a dedicated education website (ScratchEd 2013) with resources for all ages. Figure 3 shows Scratch in editing/code viewing mode.

---

<sup>129</sup> However a downloadable version of Scratch 2.0 is being developed for use where internet connectivity is limited or unavailable. The downloadable edition of the previous version (1.4) is still available to use until this is ready.

<sup>130</sup> This is now called PicoBoard – See (MIT Media Lab 2013b) and (The Playful Invention Company 2010)

<sup>131</sup> This tips tab is also where the getting started tutorial is located but in some steps it gets minimised and it isn't immediately obvious how to restore it.



**Figure 3: Scratch in editing/code viewing mode**

#### **Usability of the programming tool/environment: 4**

Usability is similar to App Inventor being focused around puzzle piece blocks yet is simpler by being completely web-based and blocks usage and design being on one screen/window<sup>132</sup>. The design is also more basic/simple and there is a tips/help screen to explain the components.

#### **Ease of use and intuitiveness of the programming language: 4.5**

Again the usage is similar to App Inventor due to the use of blocks but they are named and explained in a clearer and simpler way and more help/tips on their usage are provided.

#### **Programming concepts covered: 3**

Most concepts are included in a simple way; however advanced concepts like OOP are not featured.

<sup>132</sup> In App Inventor designing the interface is on one screen/window and adding code via blocks is on another screen/window (the blocks editor).

**Relevance to industry: 2**

Due to its simplicity and the programs created being basic and limited to the Scratch website it is unlikely to be used in industry. However it is still useful for learning programming concepts which are useful in industry.

**Ability to create real-world and relevant applications: 2**

As it is a basic teaching tool and as the outputs (programs) are basic and only exist on the Scratch website they are unlikely to be of use in the real-world or be hugely relevant. However it does offer the ability to quickly and easily demonstrate concepts.

**Interactive features: 3.5**

Scratch is primarily used to create basic games and animations but there are some more advanced interactive features such as motion detection via a webcam and plans to add integration with external devices.

**Motivational potential: 3**

Its ease of use and interactive features make it appealing, especially to younger age groups, but its problems with lack of relevance could reduce motivation to continue using it.

**Quality of documentation and amount of teaching resources available: 4.5**

There are many excellent and easy to understand tutorials and resources available as well as help/tips within the environment.

**Longevity and update frequency: 4.5**

It is developed and maintained by MIT (2013) and is regularly updated and has support to keep the project running.

**Total:** 31 (68.89%) **Average:** 3.44 **Weighted Total:** 96 (71.11%) **Weighted Average:** 3.56

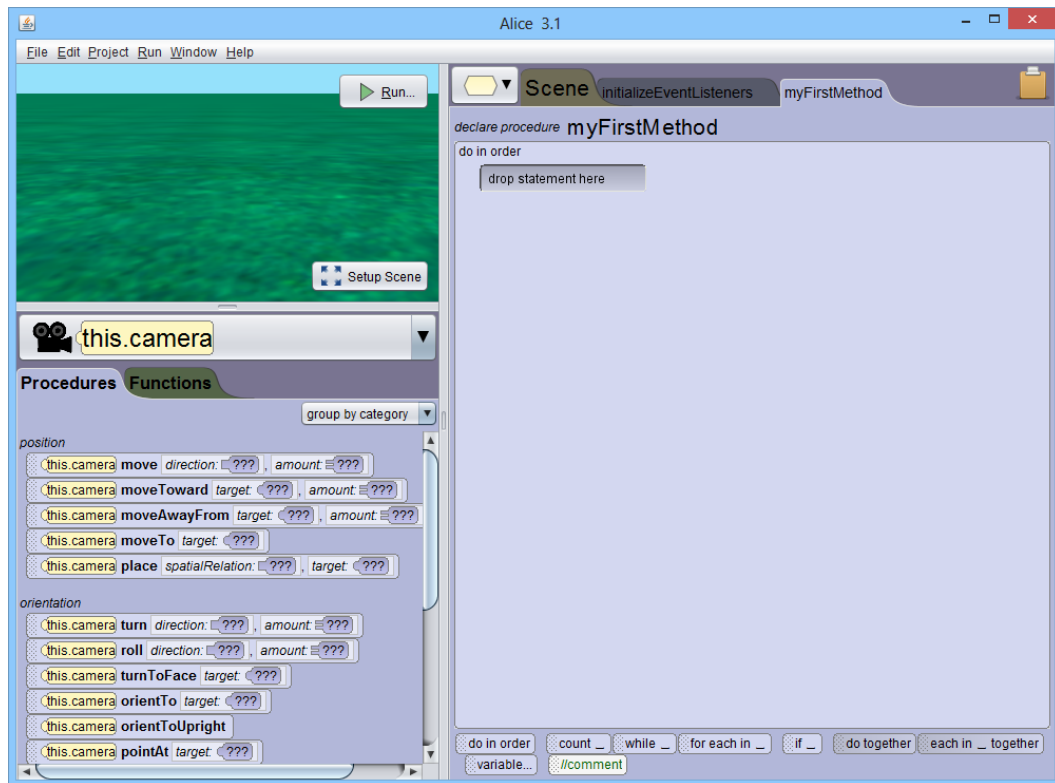
### *Alice*

Although Alice is a block-based visual tool like Scratch and App Inventor it is very different. The major difference being it aims to teach OOP; objects used (e.g. a person, a rock etc.) are instances of classes and can use procedures and functions from its parent class. For example an instance of an `AdultPerson` class can use its 'say' procedure to speak the specified text<sup>133</sup>. The use of OOP, while useful for understanding OOP concepts for moving onto more formal text-based OOP languages like Java, makes it more difficult to understand. Alice uses a 3D environment and is based around storytelling and each program is run as a sort of movie/animation. Objects are added to the scene editor and can be configured such as adjusting hair colour. Code/blocks can be added to interact with the object via the code editor e.g. make a person say "Hi" and a cat respond with "Meow". Even the most basic example uses a class for the scene, initialisation of event listeners and triggering of the `myFirstMethod` method/function to start the program. Although most of this can be skipped until later in a course<sup>134</sup> and just tell students to put the code they wish to execute in the `myFirstMethod` method/function it can still make starting using Alice daunting. Additionally although Alice uses similar drag-and-drop functionality to other visual tools like Scratch and App Inventor and shows which functions are applicable for each class and available options/values it isn't as obvious or as easy to use. The additional level of complexity while useful for teaching OOP may confuse and frustrate first-time programmers and therefore as recommended by Adams (2010) it is advisable to use a simpler system like Scratch to introduce programming concepts prior to the use of Alice. The Alice website contains many tutorials and resources for teaching Alice and they even provide a set of instructional materials including 4 units of multiple lessons which teachers can use to teach the basics of programming with Alice. The Alice environment is shown in figure 4.

---

<sup>133</sup> This is displayed as a speech bubble.

<sup>134</sup> You can leave explaining how the `myFirstMethod` method/function is executed when the application starts until the students have a greater understanding of Alice and OOP concepts.



**Figure 4: The Alice programming environment/tool**

**Usability of the programming tool/environment: 3**

The drag-and-drop system makes objects easy to create and add interactions and functionality; this and the setting of objects properties can be done without the need to write code. The interface is reasonably simple to use and see how components interact and values/options that apply to them but is more complex in comparison to Scratch and App Inventor.

**Ease of use and intuitiveness of the programming language: 2.5**

Due to the use of OOP it is quite complex; it is also not very intuitive how components are related, which options/values are applicable to functions, and so forth. Also it is hard to establish how to add things like control statements; the environment focuses on animating characters more than underlying programming concepts.

**Programming concepts covered: 4.5**

Alice uses OOP and covers almost all programming concepts.

**Relevance to industry: 4**

Although industry probably doesn't use Alice it is based around learning Java and OOP which is used extensively in industry.

**Ability to create real-world and relevant applications: 2**

As it is based around storytelling and animation the applications created offer little value other than for teaching purposes.

**Interactive features: 2**

It can make interactive stories, animations and games but it is difficult to connect to external devices, webcams and so forth and usually requires the creation of custom interfaces for them.

**Motivational potential: 3**

The interactive features and relevance to industry via the links with Java make it potentially motivating but the lack of real-world and relevant applications reduce its appeal.

**Quality of documentation and amount of teaching resources available: 3**

There is a reasonable amount of documentation and teaching resources available for Alice but these are not as extensive as some other languages/tools.

**Longevity and update frequency: 4.5**

It is developed and maintained by the Carnegie Mellon University with contribution and support from other universities and industry. It is regularly updated and has support to keep the project running.

**Total: 28.5 (63.33%) Average: 3.17 Weighted Total: 83.5 (61.85%) Weighted Average: 3.09**



## Text-based programming languages

### *Python*

Python is commonly used in industry and is used in many application domains (Python 2013a). It is a high-level general purpose dynamic programming language and is designed to have simple syntax that is easy to learn and shorter than many languages while being a powerful and effective approach to Object-Oriented Programming (Python 2013b). The simplicity and compactness is achieved via features such as:

- “high-level data types allow you to express complex operations in a single statement;
- statement grouping is done by indentation instead of beginning and ending brackets;
- no variable or argument declarations are necessary” (Python 2013c)

This approach is ideal for education as it can introduce text-based programming covering all programming concepts in a simple and intuitive way while enabling significant programs to be created. Another useful feature is extensions and modules can be created with other languages such as C or Java.

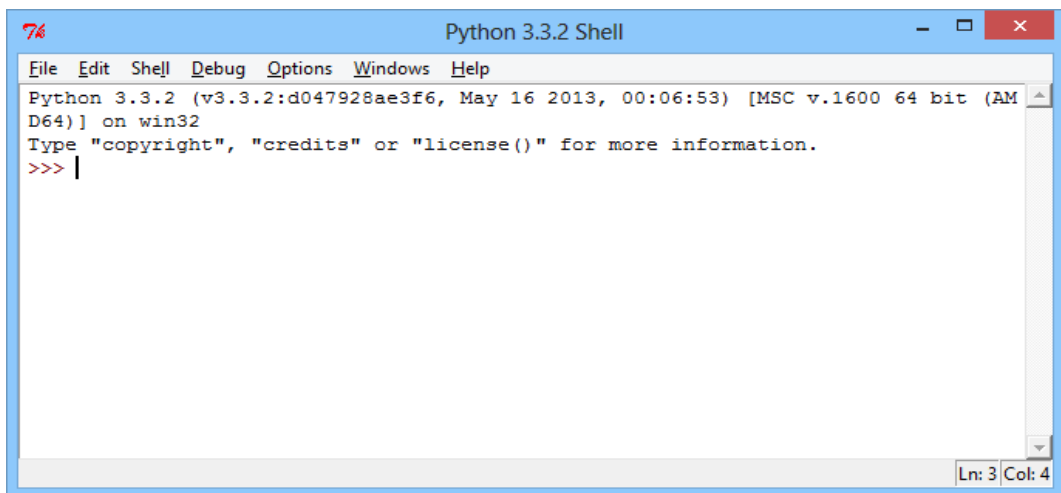
It is primarily used to create command line/console programs but there are toolkits such as Tk to create Graphical User Interfaces (GUIs).

There is plenty of documentation on the Python website including a getting started tutorial. In addition there are learning environments to help with code writing with features such as tips and code auto-completion and there is even an extension to integrate it into Visual Studio (CodePlex 2013). There is also support via a large user community and specialist groups including an education specialist group (Python 2013d) which provides information and links to resources to help educators teach Python.

Figure 5 shows the Python command line and Python’s IDLE Graphical User Interface/IDE.



**Figure 5a: Python Command Line**



**Figure 5b: Python's IDLE Graphical User Interface/IDE**

**Usability of the programming tool/environment: 2.5**

The default tools are very basic, the Graphical User Interface/IDE (IDLE) has basic auto-completion when in interactive mode but in general its features are limited. However there are other editors available with more features and the extension for using Python in Visual Studio is highly recommended.

**Ease of use and intuitiveness of the programming language: 3.5**

The language is designed to have a shorter simpler syntax than other languages and to be easy to understand and learn.

**Programming concepts covered: 4**

It includes all programming concepts in a simple and intuitive way.

**Relevance to industry: 4.5**

It is commonly used within industry.

**Ability to create real-world and relevant applications: 4.5**

It is used in many application domains with real-world and relevant uses.

**Interactive features: 4**

It can be used in many interactive ways: it can connect to electronics to for example create a robot, interact with networks via a socket interface, be used to create games, and so forth.

**Motivational potential: 4**

With its many potential uses and interactive features it has many motivational and appealing uses/examples.

**Quality of documentation and amount of teaching resources available: 4**

There is a wide variety of well written documentation and easy to follow examples available.

**Longevity and update frequency: 4**

It is open source and there is a large community dedicated to its development, maintenance and continued availability.

**Total: 35 (77.78%) Average: 3.89 Weighted Total: 102.5 (75.93%) Weighted Average: 3.8**

***Logo*****Usability of the programming tool/environment: 4.5**

It is very basic with simple commands entered into the tool/environment which are executed with output shown on the screen or via an external device such as a turtle robot.

**Ease of use and intuitiveness of the programming language: 4.5**

It is designed to be very simple and commands are logical and easy to understand.

**Programming concepts covered: 2.5**

It covers the main basic programming concepts so is an excellent simple introduction to programming.

**Relevance to industry: 1**

It is not used in industry and has little resemblance to code used in industry.

### Ability to create real-world and relevant applications: 1

Its programs are mainly for teaching and have little relevance or value to the real-world.

### Interactive features: 2

The commands are shown on the screen or via a turtle robot.

### Motivational potential: 2.5

It is an easy, fun and motivational way of introducing programming especially for young children but it is limited so it is advised to quickly move onto other more complex languages.

### Quality of documentation and amount of teaching resources available: 3

There are many versions of Logo so finding documentation for a specific version may be difficult but syntax is similar between versions. There is an online version for learning Logo at (Turtle Academy 2013) which is shown in figure 6.

### Longevity and update frequency: 2.5

Due to the age of Logo support is limited but due to its popularity and as there are many versions it should be supported for many years.

**Total: 23.5 (52.22%) Average: 2.61 Weighted Total: 77 (57.04%) Weighted Average: 2.85**

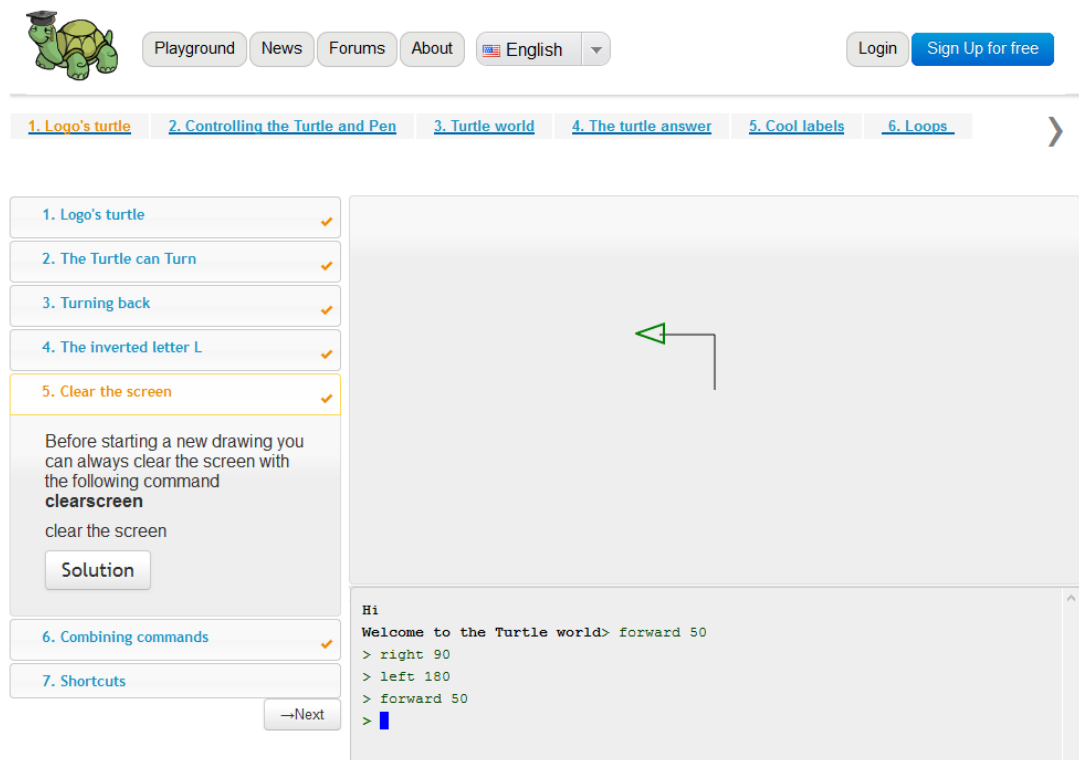


Figure 6: The Turtle Academy version of Logo

## *C# and Visual C#*

### **Usability of the programming tool/environment: 4.75**

The Visual Studio IDE (shown in figure 7) is very popular and has many features to assist programming such as highlighting of errors, auto-completion, an intellisense list that appears as the user types to assist them with the code they wish to write<sup>135</sup> and so forth.

### **Ease of use and intuitiveness of the programming language: 4.25**

Although it is a full OOP text-based language it has been designed to be easy to understand and learn.

### **Programming concepts covered: 4.5**

It includes all programming concepts in a reasonably simple and intuitive way.

### **Relevance to industry: 4.5**

Many C based languages are used in industry and C# is frequently used especially when used with ASP .NET.

### **Ability to create real-world and relevant applications: 4.5**

Due to its range of uses and ability to use all programming concepts with both command-line/console and GUI Windows applications, many relevant applications can be created.

### **Interactive features: 4.5**

Like many C based languages it can be used with multiple devices including electronics to create interactive applications.

### **Motivational potential: 4.5**

Due to its ease of use and Rapid Application Development approach, it can quickly and easily be used to create motivational examples.

### **Quality of documentation and amount of teaching resources available: 4.5**

There is a lot of help within Visual Studio to explain components (functions, parameters etc.) and many in-depth tutorials, examples, books and other resources available.

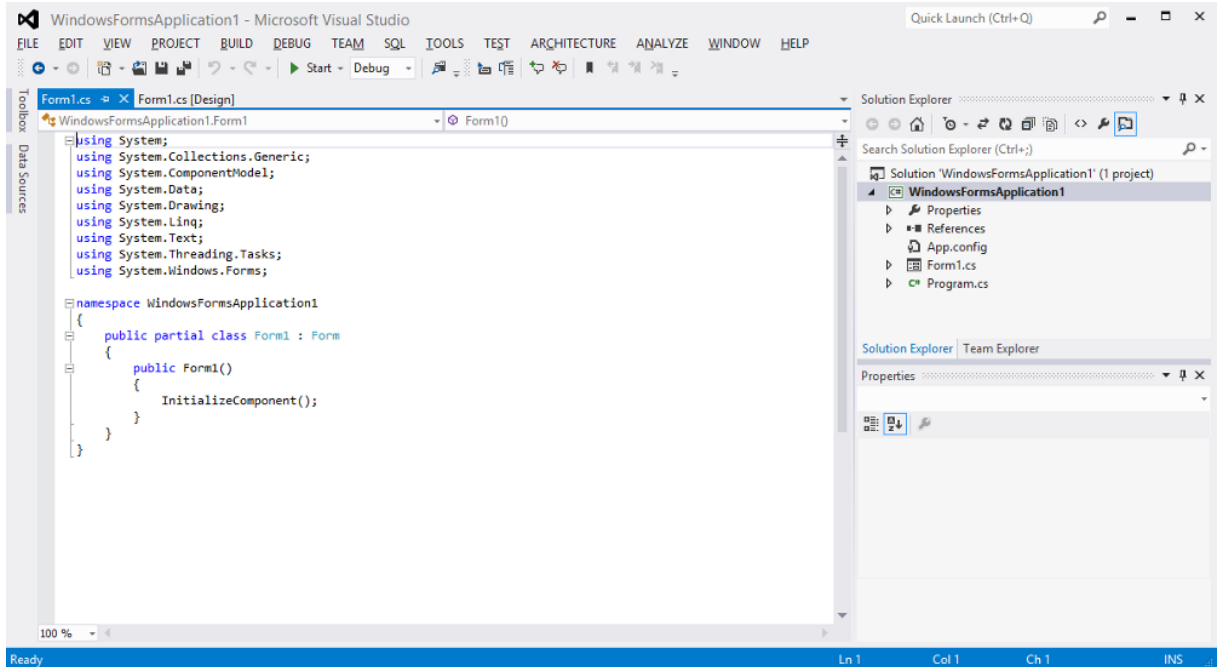
---

<sup>135</sup> The list shows possible words (functions, properties etc.) which the user can choose from rather than completing manually typing it (ideal when you aren't sure how to spell something or can't remember what it's called) along with a description of what they are/do.

**Longevity and update frequency: 4.5**

It is developed and maintained by Microsoft who invest in its future and regularly update it.

**Total: 40.5 (90%) Average: 4.5 Weighted Total: 121.5 (90%) Weighted Average: 4.5**



**Figure 7: Visual Studio**

**Visual Basic .NET**

**Usability of the programming tool/environment: 4.75**

Visual Basic .NET, like C#/Visual C#, also uses Visual Studio so has its advantages.

**Ease of use and intuitiveness of the programming language: 4**

Although it is a full object-oriented text-based language it isn't as complex as other OOP languages such as Java.

**Programming concepts covered: 4.5**

It includes all programming concepts in a reasonably simple and intuitive way.

**Relevance to industry: 3.75**

Although it is unlikely to be used to create commercial applications it is popular for creating quick prototypes due to its Rapid Application Development approach.

**Ability to create real-world and relevant applications: 4.5**

Due to its range of uses and ability to use all programming concepts with both command-line/console and GUI applications many relevant applications can be created.

**Interactive features: 4**

In a similar way to C# it can be used in interactive ways but this is less common.

**Motivational potential: 4.5**

Due to its ease of use and Rapid Application Development approach it can quickly and easily be used to create motivational examples.

**Quality of documentation and amount of teaching resources available: 4.5**

There is a lot of help within Visual Studio to explain components (functions, parameters etc.) and many in-depth tutorials, examples, books and other resources available.

**Longevity and update frequency: 4.5**

It is developed and maintained by Microsoft who invest in its future and regularly update it.

**Total:** 39 (86.67%) **Average:** 4.33 **Weighted Total:** 118 (87.41%) **Weighted Average:** 4.37

*Microsoft Small Basic***Usability of the programming tool/environment: 4**

The environment is very basic to appeal to new programmers and helps the user to code with documentation and an intellisense list.

**Ease of use and intuitiveness of the programming language: 4.5**

Its minimal 14 keyword syntax makes it easy and intuitive.

**Programming concepts covered: 4**

Although the syntax is basic it covers almost all programming concepts.

**Relevance to industry: 2**

As it is an introductory programming language it isn't used in industry but it is useful as an introduction prior to using C# or Visual Basic .NET which are used in industry. There is also a feature to convert a Small Basic program into a Visual Basic .NET program.

**Ability to create real-world and relevant applications: 1**

Its programs are usually creating for teaching/learning purposes and have little relevance or value to the real-world.

**Interactive features: 1**

There are no specific interactive features but interactive programs like games can easily be created<sup>136</sup>.

**Motivational potential: 3**

The ability to quickly create programs should motivate students and show the appeal of programming.

**Quality of documentation and amount of teaching resources available: 4**

There is a reasonable amount of quality documentation, resources and a user community available as well as external sites producing additional resources.

**Longevity and update frequency: 2**

Its longevity could be queried as the FAQ on the Small Basic website (Microsoft 2013) says their continued investment into it depends on its popularity; considering it hasn't been updated since 2011 suggests investment in it has ceased.

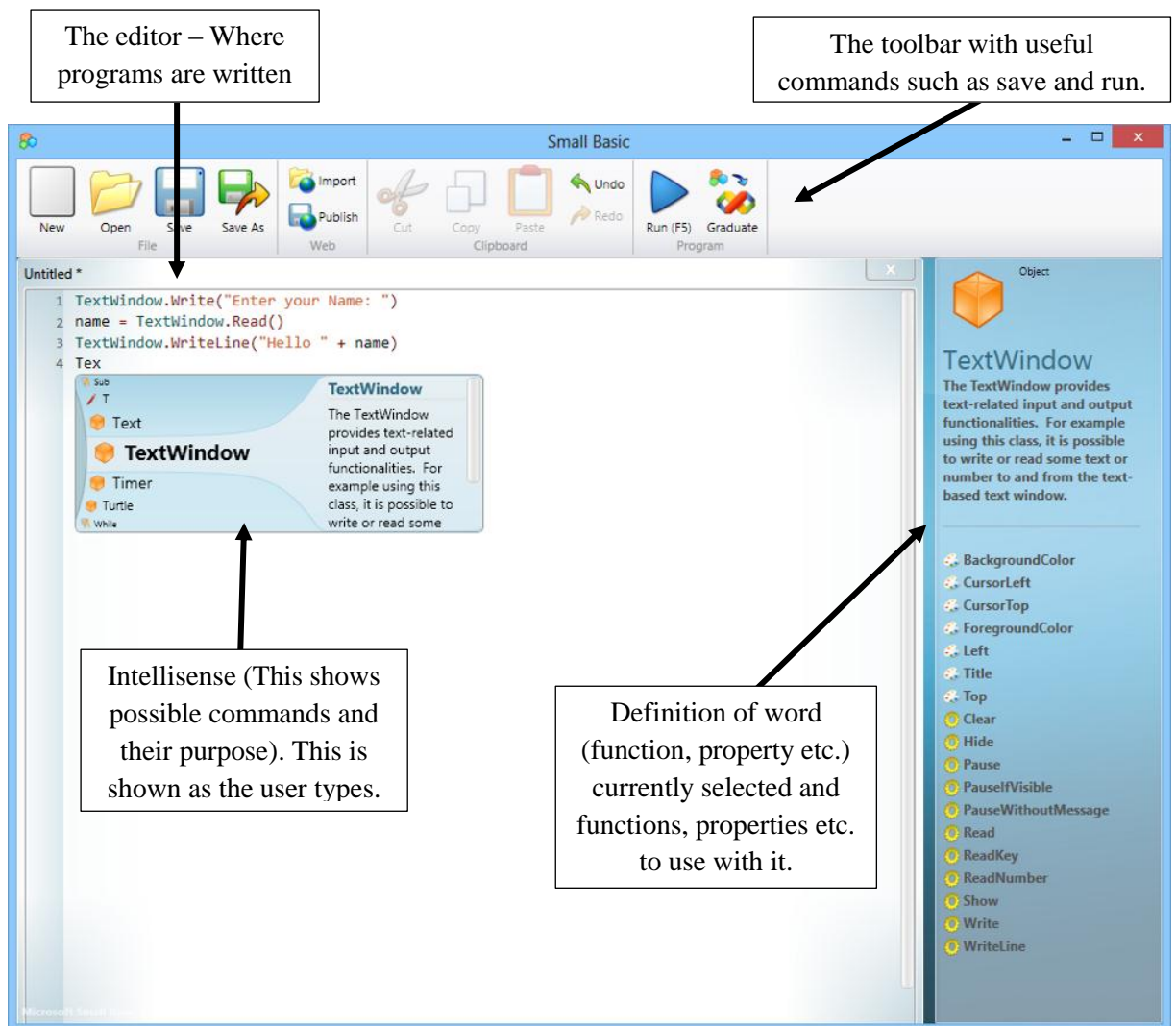
**Total: 25.5 (56.67%) Average: 2.83 Weighted Total: 84 (62.22%) Weighted Average: 3.11**

Figure 8 shows Small Basic and its main features.

---

<sup>136</sup> For example the getting started guide shows how easily a pong style game can be made.





**Figure 8: Small Basic**

### *Java*

**Usability of the programming tool/environment:** Not applicable

There is no default/official tool/environment for writing Java code; however there are many unofficial tools/environments available with varying features.

**Ease of use and intuitiveness of the programming language:** 3

As a C derivative it should be similar to users of similar languages but in general it is difficult to understand especially for novices.

**Programming concepts covered:** 4.75

It covers all programming concepts and has extensive functionality.

**Relevance to industry: 4.75**

It is extremely popular in industry and is used for many different purposes on many different devices.

**Ability to create real-world and relevant applications: 4.75**

It is used for creating many real-world and relevant applications.

**Interactive features: 4.75**

Due to its wide range of features it can be used in many interactive ways.

**Motivational potential: 2.5**

While it is powerful and can create many motivational and appealing programs its complexity may reduce its appeal especially with beginners.

**Quality of documentation and amount of teaching resources available: 4.5**

There is extensive documentation and teaching resources available.

**Longevity and update frequency: 5**

It is developed and supported by Oracle and due to its popularity and extensive use it is frequently updated and its longevity is supported.

**Total<sup>137</sup>: 34 (85%) Average: 4.25 Weighted Total<sup>138</sup>: 95 (82.61%) Weighted Average: 4.13**

---

<sup>137</sup> This is out of 40 due to usability attribute being not applicable.

<sup>138</sup> This is out of 115 due to usability attribute being not applicable.

## References

- Adams, J.C., 2010. Scratching Middle Schoolers' Creative Itch. In: *Proceedings of the 41st ACM technical symposium on Computer science education*, SIGCSE '10, 10-13 March 2010 Milwaukee. New York: ACM, 356-360. Available from: <http://dl.acm.org/citation.cfm?id=1734385> [Accessed 17 May 2013].
- CodePlex, 2013. *Python Tools for Visual Studio*. Microsoft. Available from: <http://pytools.codeplex.com> [Accessed 17 August 2013].
- Lifelong Kindergarten Group, 2013. *Lifelong Kindergarten*. Lifelong Kindergarten Group. Available from: <http://llk.media.mit.edu> [Accessed 16 August 2013].
- Microsoft, 2013. *FAQ*. Microsoft Corporation. Available from: <http://smallbasic.com/faq.aspx> [Accessed 13 June 2013].
- MIT, 2013. *Massachusetts Institute of Technology*. Massachusetts Institute of Technology. Available from: <http://web.mit.edu> [Accessed 16 August 2013].
- MIT Media Lab, 2013a. *MIT Media Lab*. MIT Media Lab. Available from: <http://www.media.mit.edu> [Accessed 16 August 2013].
- MIT Media Lab, 2013b. *Sensor Boards*. MIT Media Lab. Available from: [http://info.scratch.mit.edu/Sensor\\_Boards](http://info.scratch.mit.edu/Sensor_Boards) [Accessed 16 August 2013].
- The Playful Invention Company, 2010. *PicoBoard*. The Playful Invention Company. Available from: <http://www.picocricket.com/picoboard.html> [Accessed 16 August 2013].
- Python, 2013a. *Applications for Python*. Python. Available from: <http://www.python.org/about/apps/> [Accessed 13 June 2013].
- Python, 2013b. *The Python Tutorial*. Python. Available from: <http://docs.python.org/3/tutorial/> [Accessed 13 June 2013].
- Python, 2013c. *Whetting Your Appetite*. Python. Available from: <http://docs.python.org/3/tutorial/appetite.html> [Accessed 13 June 2013].
- Python, 2013d. *EDU-SIG: Python in Education*. Python. Available from: <http://www.python.org/community/sigs/current/edu-sig> [Accessed 13 June 2013].
- ScratchEd 2013. *ScratchEd*. MIT Media Lab. Available from: <http://scratched.media.mit.edu> [Accessed 16 August 2013].
- Turtle Academy, 2013. *Turtle Academy*. Turtle Academy. Available from: <http://turtleacademy.com> [Accessed 17 August 2013].

## Appendix 7 – Case Study: Tasks – Further details

### Part 1

#### Tasks

1. Animating a character including the usage of a loop to repeat the characters animations
2. Traffic light example:
  - Loop through a traffic lights sequence
  - Add delays to have set times for each part of the sequence

#### Challenges for advanced students/teams

1. Based on task 2 add a second set of traffic lights and link their sequences together e.g. when one is green the other is red.
2. Adjust task 1 (perhaps saving it as a different file) and make the cat meow (i.e. play its meow sound) when it is clicked on.
3. Based on task 2 with one set of traffic lights add a pedestrian crossing to make the traffic lights go to red and delay the sequence (give the pedestrian time to cross).
4. Have 2 traffic lights with their sequences linked like in the earlier challenge and have pedestrian crossings on both. Bear in mind that stopping one traffic light delays its sequence from restarting and this will make the traffic lights' sequences out of sync. Therefore it would be difficult to get them back to normal but essential if both of them being green would cause a crash.

### Part 2

#### Tasks<sup>139</sup>

1. Using a Jelly Baby as a switch to trigger the playing of a sound “Make the Jelly Baby sing”.
2. Making a LED flash on and off.
3. Adjusting the traffic light example from part 1 task 2 to make LEDs turn on and off to match the different lights in the sequence as they are shown on the screen.

---

<sup>139</sup> The first 2 tasks are Scratch equivalents to the Python examples from (OCR 2013)

### *Challenges for advanced students/teams:*

1. Add a pedestrian crossing to 1 set of traffic lights like in part 1 challenge 3 and add a paper clip switch like in part 2 task 1 to trigger the pedestrian crossing. It could perhaps use a Jelly Baby as the switch “The Jelly Baby arrives at the crossing”.
2. Put a LED into or on the head of a Jelly Baby and then create an animation so that when the Jelly Baby has a thought it is shown in a thought bubble on the screen and the LED turns on and then when the thought has ended the light goes off.

### **References**

OCR, 2013. *Raspberry Pi*. Cambridge: OCR. Available from: <http://www.ocr.org.uk/qualifications/by-subject/computing/raspberry-pi> [Accessed 17 August 2013].

## Appendix 8 – Case Study: Event timetable

See table 1 for an envisaged events timetable for the full day version of the event:

**Table 1: Event timetable**

<b>Time</b>	<b>Item</b>
9:00am	Arrivals and introduction
9:20am	Part 1
11:00am	Break
11:15am	Introduction to the electronics used in part 2
11:25am	Part 2
12:25pm	Lunch break
1:10pm	Continue with part 2
2:10pm	Survey
2:45pm	Conclusion
3:00pm	Depart

All timings are approximate. Some schools may wish to have a slightly shorter day to allow for time to travel from the school to the location and return within the hours of the school day<sup>140</sup>. Equally a school may prefer a half day event; the variety of tasks and challenges allows for the days content to be reduced if required. Additionally the skills of the students attending can be assessed prior to the event and it may be feasible to skip some of the basic tasks<sup>141</sup> and thus reducing time required to complete the events content. In this case it may be necessary to provide completed programs for the basic tasks that were skipped, as the later tasks are based on some of these (ordinarily when running the full event the students would have created these). Also the survey questions relating to the event may need to be adjusted to take into account the reduced content<sup>142</sup>.

---

<sup>140</sup> The school can request that the event is run at their school to simplify administration tasks such as getting permission for students to go on a trip out of the school. In this situation travel time can be taken out of the timetabling considerations.

<sup>141</sup> Students may already have enough basic Scratch knowledge that they do not need to do these tasks.

<sup>142</sup> For example if you skip part 1 you can't ask the question asking about it.

## Appendix 9 – Case Study: Tasks Worksheets/Hand-outs

### Scratch introduction

If you are unfamiliar with Scratch or want reminding of the basics here is a simple introduction; if you are happy with using Scratch you can go straight onto the tasks in part 1.

When you open Scratch you will see a sprite (character/object you can interact with) of a cat (there are also others you can choose from).



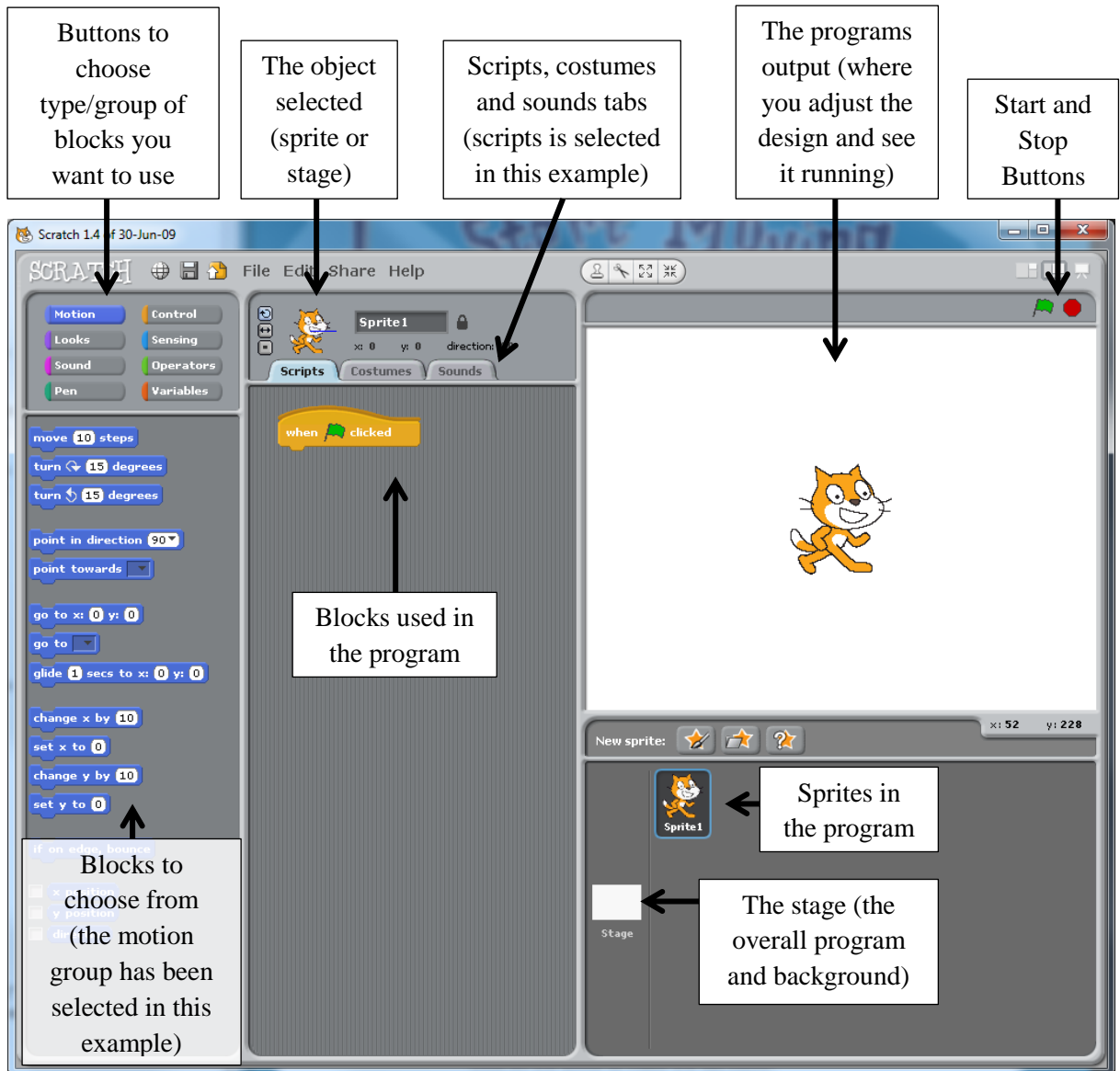
This cat sprite has 2 costumes (see the costume tab) which are different images of the character, for this sprite these show 2 different walking positions. Sprites can also have sounds attached to them (see the sounds tab) and the cat has one sound “meow”. Sprites also have a scripts tab which is where you can add blocks/code to interact with it. Therefore we have a character we can interact with that has different appearances/looks and set sounds it can make; you can add more costumes and sounds if you wish.

There are blocks/puzzle pieces for various programming components to allow you to add functionality to your program (animate your sprite, play sounds, perform calculations etc.); these are the equivalent of writing code in more complex programming languages. The blocks are colour coded by type and can be chosen by clicking the type buttons in the top left. Blocks lock together like puzzle pieces and only snap together with components/blocks that work together so there is no need to worry about getting things wrong.

Blocks can either be added in a sprites script tab for blocks/code relating to the sprite or to the stage which is for the background and overall program.

The window in the top right shows the program’s output, you can adjust the design such as modifying the sprites position, size and so forth. This is also where the program is run; it is started by clicking the green flag and stopped by clicking the red stop sign, although alternatives can be set in the code/blocks such as start when space bar is clicked.

Figure 1 shows the Scratch program/tool and the features mentioned. If you need more assistance on getting started with Scratch they provide an excellent getting started guide at <http://scratched.media.mit.edu/sites/default/files/GettingStartedGuidev14.pdf>.



**Figure 1: The Scratch program/tool and its main features.**











**Note:** Solutions to problems you may come across in a task are below its instructions. If you need any other help please ask.

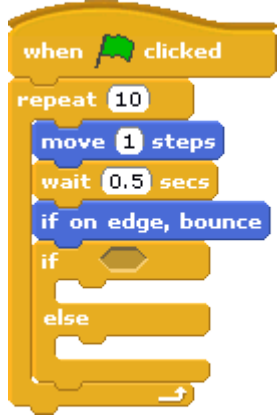


## Part 1

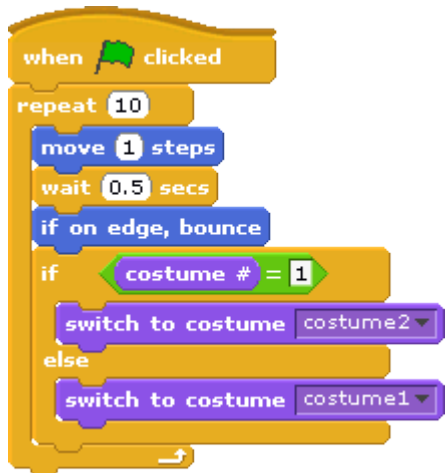
### Task 1 of 2: Animating a character

In this task you will learn how to animate a character so it walks across the screen.

Step	Details	Related Images
1	<p>Start a new Scratch program/file (starting Scratch will do this automatically).</p> <p>We will be animating the cat sprite/character which you will see on the screen (it is the default sprite/character for every new program).</p> <p>The cat should already be selected, if not select the cat (Sprite1) by clicking on its icon in the sprites area (bottom right).</p>	
2	<p>Add the 'when green flag clicked' control block (it is in the control group) which performs actions when the green flag is pressed.</p>	
3	<p>By connecting a block/s to another block they happen after that blocks actions are completed i.e. in sequence (the order specified). For example by connecting block A to block B you are saying do A then B.</p> <p>Therefore a block added to the 'when green flag clicked' block will happen after the green flag is clicked. Connect a 'move 10 steps' block which moves the sprite forward a set number of steps (10 in this example).</p> <p>Tip: You can click on the 10 and change it for any number to modify the amount of steps moved.</p>	
<p><b>Try it out/test it:</b> Let's test it - Click the green flag and see the cat move.</p>		

<p><b>4</b></p>	<p>The cat's walking isn't very realistic as we don't see the steps, however if we introduce a delay between each step it will look more real.</p> <p>First disconnect the 'move 10 steps' block.</p> <p>Connect a 'repeat 10' control block to the 'when green flag clicked' block. The 'repeat 10' block is a loop which has been set to repeat its contents (the block inside it) 10 times. You can change the 10 to any number to adjust the number of times it repeats.</p>	 <p>Note: The 'move 10 steps' block has been disconnected but still remains as we will need it later.</p>
<p><b>5</b></p>	<p>Next change the steps in the 'move 10 steps' block to 1 and put it in 'repeat 10' loop. Now 1 step is repeated 10 times; this may seem unnecessary when we can just say move 10 steps but it allows us to add a delay between each step for more realistic walking.</p>	
<p><b>6</b></p>	<p>To add a delay between each step add a 'wait 1 secs' control block below the 'move 1 steps' block. The 'wait 1 secs' block adds a 1 second delay thus creating a gap between each step.</p>	
<p><b>Try it out/test it:</b> Now run the program (click the green flag) and see the improved walking.</p>		
<p><b>7</b></p>	<p>The walking seems a bit slow so if we reduce the time in the wait block 'wait 1 secs' (the delay) the walking will be quicker (0.5 seconds seems more realistic). Click on the 1 and change it to 0.5</p>	
<p><b>8</b></p>	<p>After running the program a few times the cat will walk past the edge of the screen and we will lose it. To solve this problem add a 'if on edge, bounce' motion block which turns the sprite around when it hits the edge of the screen. A good place to add this is at the bottom of the repeat 10 loop.</p> <p>Unfortunately the bounce turns the cat upside down and solving this problem is a more advanced challenge which you can try if you have time later.</p>	
<p><b>Try it out/test it:</b> Run the program and see the faster walking, if you run it many times the cat will hit the edge of the screen and then turn around and start walking the other way.</p>		

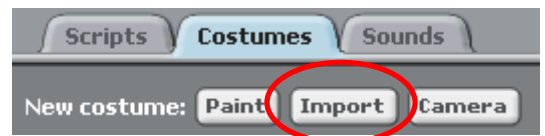
<p><b>9</b></p>	<p>Superb, we now have a walking cat. It would however be better if its legs moved as it walked. Fortunately the cat has another costume of a different walking position so if we keep switching the costumes with each step it will look like the cat is walking.</p> <p>To alternate between the cats costumes we will use an 'if else' control block/statement. This says if a condition is true do these blocks, if not do some other blocks.</p> <p>The default costume is numbered 1 and the other costume is numbered 2. Therefore we can say if costume number is 1 change costume to costume 2 else (i.e. it is already 2) change it to costume 1.</p> <p>First we will add the 'if else' block inside the repeat loop at the bottom.</p>	
<p><b>10</b></p>	<p>Now we need to set the condition for the 'if else' block to check. We will take the 'equals comparison' operator block which looks like this  and has boxes for adding items to compare i.e. if the left part is the same as the right part then it is true, if not it is false.</p> <p>We need to compare the costume number to see if it is costume 1. To do this drag the 'costume #' looks block into the left box of the 'equals comparison' block and add a 1 in the right box (you can click in the box and type in 1) to compare it against. See image on the right.</p> <p>Note: # means number.</p> <p>The 'costume #' block is a bit like a variable and it stores the number of the costume the sprite is currently using. Therefore we can use this to work out which costume to switch/change to.</p>	

<p><b>11</b></p>	<p>Now if the costume is numbered 1 it will run/use the block(s) connected to the ‘if’ part, if not it will run/use the block(s) connected to the ‘else’ part. We will use the ‘switch to costume’ looks block and set the costume accordingly using the block’s drop down list/menu (if costume 1 change to costume 2 and vice versa).</p> <p>That is all you need, we have an animated walking cat. The completed blocks you should now have are shown on the right. It may be basic but has introduced many fundamental programming concepts.</p> <p><b>Remember to save it.</b></p>	
<p><b>Try it out/test it:</b> Run it and see the completed program.</p>		

### Troubleshooting problems


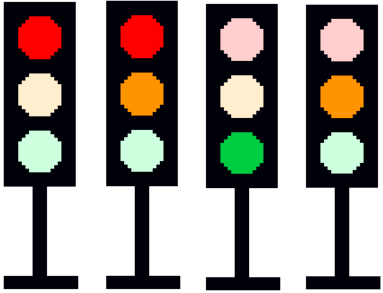



#### My cat sprite only has 1 costume

If your cat sprite only has 1 costume rather than the 2 required you will need to add the other costume. To do this go into the costumes tab and click on the import button which is one of the new costume buttons. Then find the missing costume in the animals costume folder, click on it and then press the ok button.



### Task 2 of 2: Traffic Lights

In this task you will learn how to animate a traffic light.

Step	Details	Related Images
1	<p>Start a new project (choose new from the file menu).</p> <p>Delete the cat sprite by right clicking on it and choosing delete.</p> <p>Next in the new sprite buttons click the ‘choose new sprite from file’ button and find the traffic lights sprite in the transportation folder (if you can’t see this folder you may not be in the costumes folder so click on the costumes button on the left).</p> <p>This sprite has 4 costumes for the 4 states of a traffic light sequence (red, red and orange, green and orange).</p>	 
2	<p>Just like before start by adding the ‘when green flag clicked’ control block.</p>	
3	<p>Next we will add a ‘forever’ loop (the ‘forever’ control block) which is similar to the repeat block we used earlier but without the specified amount of times it will run; it will run forever unless it is stopped via blocks/code or the stop button is used.</p>	
4	<p>As we saw in task 1 we can wait/pause for a set amount of seconds and we can also change/switch the costume of a sprite.</p> <p>Therefore you can add waits/delays and switch costumes into the forever loop to change the costumes of the traffic light sprite to show a traffic light sequence (each costume is a phase of the sequence i.e. different lights are on and off) and delays to separate them; set the wait times to whatever you feel is appropriate for the traffic light sequence.</p> <p>The completed sequence is on the right.</p> <p><b>Don’t forget to save it</b></p>	
End	<p>You should now have a working traffic light which goes through the sequence of lights and loops/repeats until it is stopped.</p>	
<p style="text-align: center;"><b>Try it out/test it:</b> Run it and see the completed program.</p>		

### Some challenges

If you want to do more tasks there are some fun challenges you can try, please ask for the part 1 challenges worksheet.

### Part 2 – Fun with the Raspberry Pi

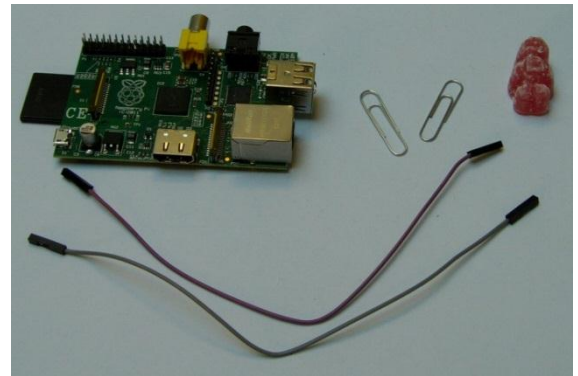
In this part we will be using the GPIO (General Purpose Input/Output) ports on the Raspberry Pi to use some electronics as inputs and outputs of our programs.

#### Task 1 of 3 – Making a Jelly Baby Sing







In this task we will use a Jelly Baby as a switch to trigger the playing of a sound and displaying of a message to make the Jelly Baby ‘sing’.

#### You will need:

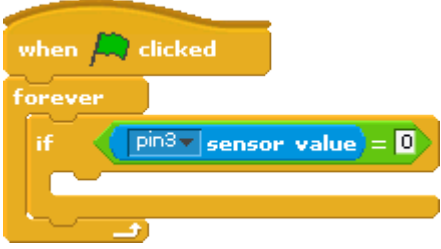

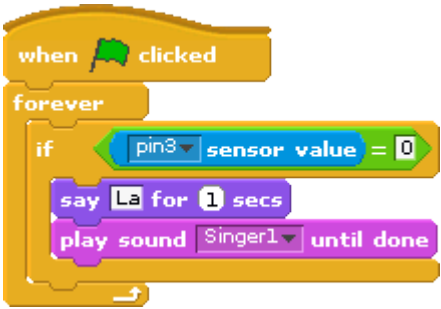
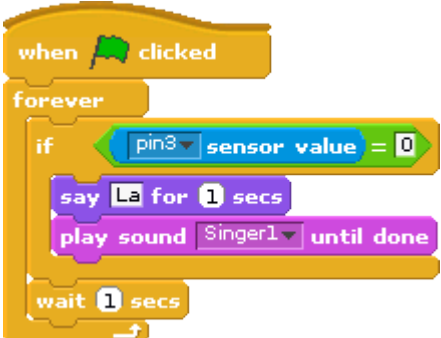
- 2x Jumper Wires (female to female)
- 2x Paper Clips (small, thin and non-plastic-coated)
- 1x Jelly Baby
- A Raspberry Pi
- Raspberry Pi accessories (keyboard, mouse, screen etc.) – Not photographed
- 1x Headphones or Speakers - Not photographed




Step	Details	Related Images
1	<p>Connect a jumper wire into pin 3 (an input/output pin which we will be using as an input) and another to pin 6 (ground).</p> <p>Tip: Pin 1 is marked on the Raspberry Pi as P1 and pins along that row are odd numbers and the other row is even numbers.</p>	
2	<p>Unbend 2 paper clips until they are straight and place 1 in the other/empty end of each jumper wire.</p> <p>If the 2 paper clips touch together they form a circuit and can be used as a very basic button/switch.</p>	

<p><b>3</b></p>	<p>Open ScratchGPIO2 (not the regular Scratch), you will find a shortcut for it on the desktop. This runs a program to handle the GPIO communications within Scratch, loads Scratch and enables Scratch's Remote Sensor Connections (RSC). This allows us to make use of the GPIO pins.</p> <p><b>Important: Do a save as to save the current file as a different name (e.g. Jelly Baby) to stop the default template being overwritten. Do not do file new as this doesn't keep the remote sensor connections enabled.</b></p> <p>First delete the cat sprite by right clicking on it and choosing delete. Next in the new sprite buttons area click the 'choose new sprite from file' button and locate the Jelly Baby in the things folder (if you can't see this folder you may not be in the costumes folder so click on the costumes button on the left). I apologise for its poor quality, my design skills aren't great.</p>	 
<p><b>4</b></p>	<p>Just like before start by adding the 'when green flag clicked' control block.</p>	
<p><b>5</b></p>	<p>Next we will add a 'forever' loop (a 'forever' control block) which will run forever unless it is stopped via code/blocks or the stop button is used. This is important so we can constantly check if any inputs happen (use of the paper clip button/switch).</p>	
<p><b>6</b></p>	<p>Now add an 'if' block inside the forever block, we will use this to check for an input (the paper clips are touching and a circuit is made/completed).</p> <p>Note: The 'if' block is like an 'if else' block only without the else part.</p>	
<p><b>7</b></p>	<p>To get/detect an input (the paper clips are making a connection like a switch) we use the 'sensor value' sensing block and set the drop down menu/list to pin3 as this is the pin we are using.</p>	<p>We will be using:</p> 



<p><b>8</b></p>	<p>We will now set the condition for the ‘if’ block. We will use the ‘equals comparison’ operator block and compare the pin 3 sensor value (the input). When there is no input it is 1 (i.e. the paper clips aren’t touching) and 0 when there is a connection (i.e. the paper clips are touching and a circuit is formed). Therefore in our ‘if’ comparison if we check that the sensor value of pin 3 equals 0 we are checking if the circuit is complete. We can then add code/blocks into the ‘if’ block that will happen when the input occurs (the circuit is complete/the paper clips are connected).</p> <p>Note: The behaviour of the pins is unusual compared to usual programming logic which has 1 as true and 0 as false; the pins values are opposite to this (also known as negative logic).</p>	
<p><b>9</b></p>	<p>Now we will make the Jelly Baby sing. In the ‘if’ add a ‘say Hello! for 2 secs’ looks block which displays a speech bubble with the message specified (Hello!) for the time specified (2 seconds). Next change the values so it is ‘say La for 1 secs’ (you can adjust the time and message to something else if you wish).</p> <p>Next we will add the playing of a sound. The sounds a sprite can play are those set in its sounds tab. There are no sounds for the Jelly Baby sprite so go into its sounds tab, click import and choose Singer1 (this is in the vocals folder) or similar and then click ok. Next return to the scripts tab and below the say block add a ‘play sound until done’ sound block; you can use the drop down menu/list to choose the sound played from the sounds your sprite has (e.g. Singer1).</p>	 
<p><b>10</b></p>	<p>It is a good idea to put a small delay in between checks for inputs so it can finish the last check before starting the next and slightly improving accuracy. Simply add a ‘wait 1 secs’ control block inside the forever loop below the ‘if’ block.</p>	



<b>Try it out/test it:</b> Run the program and then put the paper clips together completing the circuit and you will see the message and hear the sound.	
<b>11</b>	<p>Now we know our paper clip button/switch works push the paper clips into the Jelly Baby so they are close to each other but not connected/touching. Then when you squeeze the Jelly Baby the connection will be made and the message will appear and the sound file will play (the Jelly Baby “sings” when squeezed). It may take a few attempts to get the paper clips positioned correctly. You need to put them close but ensure they are not touching. Then squeezing the Jelly Baby will temporarily connect the paper clips and then when you release the pressure the paper clips spring back to open/not connected/not touching.</p> <p>Tip: You may find it useful to output the sensor value so you can see what the value currently is; this can be done by having a ‘sensor value’ sensing block replacing “hello!” in a ‘say Hello! for 2 secs’ looks block or replacing the “Hmm...” in a ‘think Hmm... for 2 secs’ looks block (I would recommend changing them to 1 secs to match the wait time), the block should be added at the bottom or top of the forever loop.</p>
	
<b>End</b>	That’s our first adventure into the world of GPIO completed. We have seen how to create a little switch from wires, paper clips and a Jelly Baby!
<b>Try it out/test it:</b> Run it and see the completed program.	

### Troubleshooting problems

#### The pin isn’t listed in the ‘sensor value’ block’s drop down menu/list

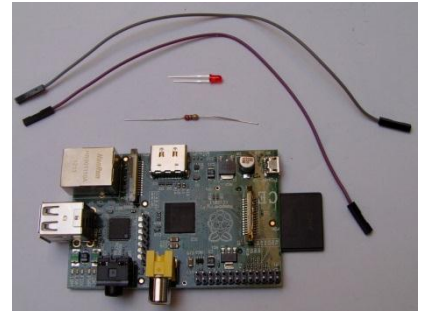
If pins are not listed in the ‘sensor value’ block’s drop down menu/list this probably means the Remote Sensor Connections are not enabled or didn’t initialise properly. To solve this right click on a ‘sensor value’ block and choose “enable remote sensor connections”; if however it is already enabled then try disabling it and re-enabling it.

**Task 2 of 3: Making an LED flash on and off**





We have seen how inputs work so we will now look at outputs. First we will make an LED flash on and off.

**You will need:**

- 2x Jumper Wires (female to female)
- 1x 220 ohm resistor
- 1x LED
- A Raspberry Pi
- Raspberry Pi accessories (keyboard, mouse, screen etc.) – Not pictured



Step	Details	Related Images
1	<p>Connect a jumper wire to pin 3 (an input/output pin which we will be using as an output) and another to pin 6 (ground). Whereas pin 3 was previously used for an input as part of the switch we are now using it for an output. This is possible because GPIO pins are general purpose and can be used as either an input or an output.</p>	
2	<p>Next take an LED and a 220 ohm resistor and wrap one end of the resistor around the shortest leg of the LED so that they are connected.</p>	
3	<p>Connect the other end of the resistor (the end that isn't connected to the LED) into the jumper wire that goes to pin 6 (ground). Then connect the jumper wire that goes to pin 3 to the other leg of the LED.</p>	
4	<p>In ScratchGPIO2 open a new file (select new from the file menu).</p>	

<p><b>5</b></p>	<p>To turn on the LED we turn on the input/output pin it is connected to (we give the pin power), likewise turning the pin off removes the power and therefore turns off the LED.</p> <p>We do this with a broadcast message which is sent via a broadcast block and tells the pin what to do (turn on or off). Therefore:</p> <ul style="list-style-type: none"> <li>• Add a broadcast block (part of the control group)</li> <li>• You can choose messages you have used before from the blocks drop down menu/list. However as we haven't used any messages before we need to add one. To add a new message click on the 'new...' option in broadcast block's drop down menu/list. Then add the message pin3on and press ok.</li> </ul> <p>Now when the block is used it turns on pin 3 and therefore turns on the LED.</p> <p>Now add another broadcast block and add a new message of pin3off. This block turns off pin 3 and therefore turns off the LED.</p>	<p>Turn pin 3 on:</p>  <p>Turn pin 3 off:</p> 
<p><b>6</b></p>	<p>Now we can turn the light on and off but we will need to put delays between the on and off otherwise it will be too quick for us to see it. Therefore add a 'wait 1 secs' control block after the 'broadcast pin3on' block and then connect the 'broadcast pin3off' block onto it. Then add on another 'wait 1 secs' control block onto the end/bottom (you can change the times if you wish). Then to complete the program connect the blocks onto a 'when green flag clicked' block so we can start the program/sequence.</p>	
<p><b>7</b></p>	<p>To keep it flashing on and off you can add a 'forever' loop.</p>	
<p><b>End</b></p>	<p>That is our first output example completed.</p>	
<p><b>Try it out/test it:</b> Run it and see the completed program.</p>		

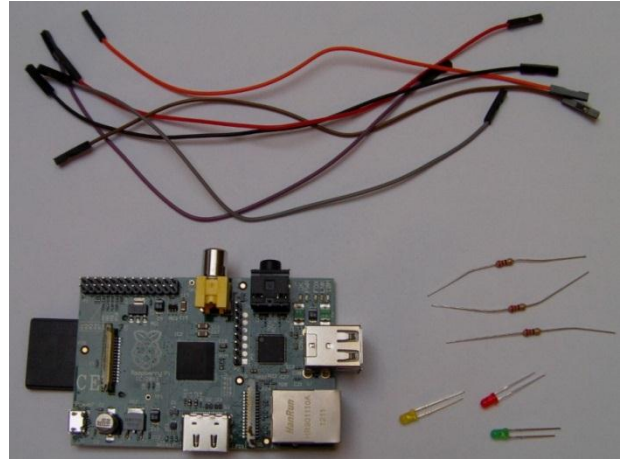
Note: GPIO outputs usually use high for on and low for off (referring to power levels) but ScratchGPIO2 has made it simpler for us by allowing us to use on/off (however you can use high or low if you wish).

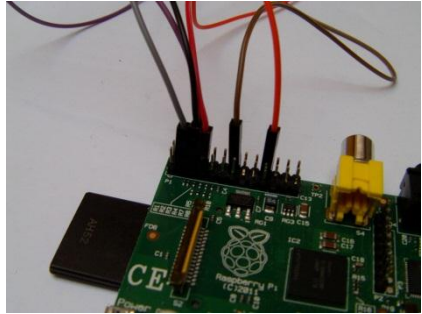

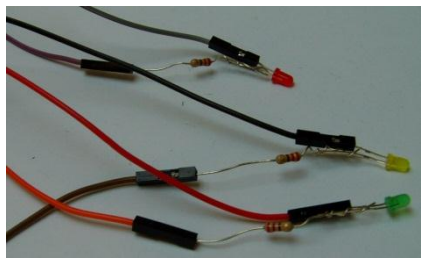
### Task 3 of 3: Traffic Lights


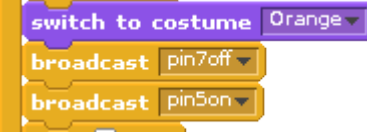
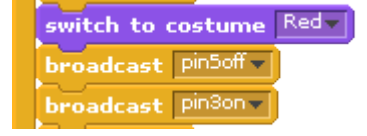

Now we will take our traffic light from part 1 task 2 and make LEDs turn on and off to match the different lights in the sequence as it is shown on the screen.

#### You will need:

- 6x Jumper Wires (female to female)
- 3x 220 ohm resistors
- 1x Red LED
- 1x Orange or Yellow LED
- 1x Green LED
- A Raspberry Pi
- Raspberry Pi accessories (keyboard, mouse, screen etc.) – Not pictured



Step	Details	Related Images
1	Connect jumper wires to pins 3, 5, 7 (input/output pins which we will be using as outputs) and 6, 14 and 20 (ground pins).	
2	Next take an LED and a 220 ohm resistor and wrap one end of the resistor around the shortest leg of the red LED so that they are connected. Repeat the process with the orange/yellow and green LEDs.	
3	Connect the red LED by connecting the other/unused end of the resistor it uses (the end that isn't connected to the LED) into the jumper wire that is connected to pin 3 and connect the other/unused leg of the LED into the jumper wire connected to pin 6. Then connect the orange/yellow LED with the other/unused end of its resistor into pin 5 and the other/unused leg of the LED into pin 14 (again via jumper wires). Finally connect the green LED with the other/unused end of its resistor into pin 7 and the other/unused leg of the LED into pin 20 (again via jumper wires).	
4	Now open the traffic light program from part 1 task 2 and use 'save as' to save it as a different file name (copy it to a new file).	

<p><b>5</b></p>	<p>Below the ‘switch costume to green’ block we will add broadcasts to turn off the red and orange/yellow LEDs and turn on the green LED.</p> <p>Just like we did in task 2 we add a broadcast block, select ‘new...’ from its drop down menu/list and add the message pin3off and press ok. This turns off the pin the red LED is using.</p> <p>Then repeat the process with another block for turning pin 5 off (the orange/yellow LED) with the message pin5off.</p> <p>Finally turn on the green LED by turning pin 7 on with another block with the message pin7on.</p>	
<p><b>6</b></p>	<p>Next create more broadcast blocks below the ‘switch to costume Orange’ block to deal with the orange light and thus turn the green LED off ‘pin7off’ and turn the orange/yellow LED on ‘pin5on’.</p>	
<p><b>7</b></p>	<p>Next we will handle the change to the red light thus placing broadcast blocks under the ‘switch to costume Red’ block with messages for pin5off (turn orange/yellow LED off) and pin3on (turn the red LED on).</p>	
<p><b>8</b></p>	<p>And finally we deal with the change to RedOrange (where red and orange are shown prior to the lights going green). As we already have red on and still need it to be on, all we have to do is add a block below the ‘switch to costume RedOrange’ block of a broadcast for turning pin 5 on to turn on the orange/yellow LED.</p>	

<p><b>End</b></p>	<p>That is our second output example completed. Run it and you will see the LEDs turn on and off to match the animations on the screen. You could make it a little more realistic by creating a traffic light model out of cardboard, Lego, wood etc. to hold the LEDs in place. Here are what the completed blocks look like:</p>	
<p><b>Try it out/test it:</b> Run it and see the completed program.</p>		

*Some challenges*

If you want to do more tasks there are some fun challenges you can try, please ask for the part 2 challenges worksheet.

## Appendix 10 – Case Study: Challenges Worksheets/Hand-outs

These challenges can be handed out to the students once they have completed the tasks for the related part. The reasoning for keeping them separate is that although these challenges are optional having them on the tasks worksheet may overwhelm and disengage the students due to the amount of content.

### Part 1 challenges

Here are some challenges for you to try if you have spare time in this part of the event. You may find for some challenges that you can save time by basing the challenge on previous tasks which will help reduce the need to repeat work. You can do a save as (to avoid losing the previous work) or export and import sprites.

#### See if you can:

1. Based on task 2 add a second set of traffic lights and link their sequences together e.g. when one is green the other is red.
2. Adjust task 1 (perhaps saving it as a different file) and make the cat meow (i.e. play its meow sound) when it is clicked on. Don't worry if you can't work out how to do this now, we will cover sounds later so you can come back to this task after that and finish it off.  
Tip: There is a 'play sound until done' sound block and a control block for detecting when a sprite has been clicked on.

#### And an extra difficult challenge:

3. Based on task 2 with one set of traffic lights add a pedestrian crossing to make the traffic lights go to red and delay the sequence (give the pedestrian time to cross). You could activate the blocks for the pedestrian crossing when you click on the traffic light (there is a control block for detecting when a sprite has been clicked on) or perhaps show a character such as the cat from task 1 arriving at the traffic light which activates the pedestrian crossing blocks (Tip: You could export the sprite from task 1 and import it into this project/challenge to avoid the need to re-add the blocks to animate the character). One thing to consider to add extra realism is the traffic light would wait until it is on red before delaying the sequence for the pedestrian to go across (they shouldn't cross when the traffic light is green). Tip: Investigate variables; I know we haven't covered these but see what you can work out and ask for help if required.

Also pedestrian crossing lights would be good.

#### And for those who want an even more difficult challenge:

4. Have 2 traffic lights with their sequences linked like in challenge 1 and have pedestrian crossings on both. Bear in mind that stopping one traffic light delays its sequence from restarting and this will make the traffic lights' sequences out of sync. Therefore it would be difficult to get them back to normal but essential if both of them being green would cause a crash. Tip: Again variables are useful here and perhaps some 'if' or 'if else' blocks.



## Part 2 challenges

Here are some challenges for you to try if you have spare time in this part of the event. You may find for some challenges that you can save time by basing the challenge on previous tasks which will help reduce the need to repeat work. You can do a save as (to avoid losing the previous work) or export and import sprites.

### See if you can:

1. Add a pedestrian crossing to 1 set of traffic lights like in part 1 challenge 3 (copy it if you wish) and add a paper clip switch like in part 2 task 1 to trigger the pedestrian crossing. It could perhaps use a Jelly Baby as the switch “The Jelly Baby arrives at the crossing”.
2. Put a LED into or on the head of a Jelly Baby and then create an animation (delete the Cat sprite and add a Jelly Baby sprite) so that when the Jelly Baby has a thought it is shown in a thought bubble on the screen and the LED turns on and then when the thought has ended the light goes off (Tip: There is a think block that shows a thought in a thought bubble).

You may ask for another Jelly Baby if required.

### *Troubleshooting problems*

#### **The pin isn't listed in the 'sensor value' block's drop down menu/list**

If pins are not listed in the 'sensor value' block's drop down menu/list this probably means the Remote Sensor Connections are not enabled or didn't initialise properly. To solve this right click on a 'sensor value' block and choose “enable remote sensor connections”; if however it is already enabled then try disabling it and re-enabling it.



## Appendix 11 – Case Study: Advice for event staff (people who will help run the event)



There will be an explanation before each part to provide introductions to the part and cover areas the students may struggle with or may not have previous experience with; for example the basics of using Scratch and the basics of using electronics with Scratch. There are challenges for each part for students who finish within the time allocated to move on to. They are designed to be a challenge and allow students to demonstrate what they have learned and to be flexible to allow for multiple approaches to be taken and for students to demonstrate creativity and intuitiveness. Although it is designed to be a challenge, the students may still need some assistance, possible solutions to the challenges are below. There is also some information for troubleshooting possible problems.

### Part 1

#### Challenge 1

Based on task 2 add a second set of traffic lights and link their sequences together e.g. when one is green the other is red.

This should be reasonably easy for the students; all they need to do is duplicate the traffic light sprite and adjust the sequence. The blocks required are:

Traffic Light Sprite 1	Traffic Lights Sprite 2
	
<p>Blocks are the same as part 1 task 2</p>	<p>This starts the sequence from green so it complements the other traffic light's sequence (when one is red the other is green and vice versa).</p>

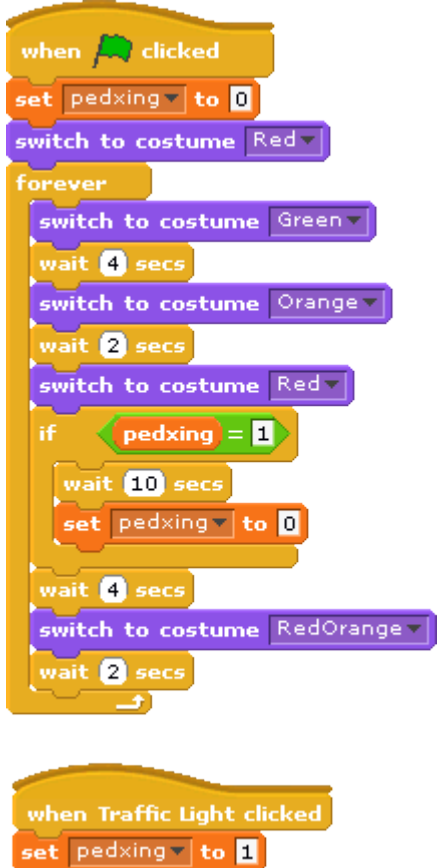
#### Challenge 2

Adjust task 1 (perhaps saving it as a different file) and make the cat meow (i.e. play its meow sound) when it is clicked on.

This one is really simple; all they need to do is add these blocks to the cat sprite's scripts tab:



### Challenge 3

Details	Blocks
<p>Based on task 2 with one set of traffic lights add a pedestrian crossing to make the traffic lights go to red and delay the sequence (give the pedestrian time to cross).</p> <p>The key to solving this problem is the use of a variable to store whether the pedestrian is crossing (pedxing). When the program is started the pedxing variable (pedestrian is crossing) is set to 0 for false. Then when the traffic light sprite is clicked on set the pedxing variable to 1 for true. Then to provide time for the pedestrian to cross we check if pedxing is 1, using an 'if' statement/block, when the sequence gets to the red light. If it is 1 we add a wait for 10 seconds (the delay to let the pedestrian cross) and then set the pedxing variable to 0 as the pedestrian crossing has finished.</p> <p>Note: The delay time can be adjusted if you wish. You could also have other ways of triggering the pedestrian crossing such as using a different sprite (having a button sprite to click, having an animation of a sprite like the cat activating the pedestrian crossing, and so forth).</p>	 <p>The image shows two Scratch code snippets. The first snippet is a 'when clicked' event block followed by a 'set pedxing to 0' block and a 'switch to costume Red' block. Below these is a 'forever' loop containing: 'switch to costume Green', 'wait 4 secs', 'switch to costume Orange', 'wait 2 secs', 'switch to costume Red', an 'if pedxing = 1' block (which contains 'wait 10 secs' and 'set pedxing to 0'), 'wait 4 secs', and 'switch to costume RedOrange'. The second snippet is a 'when Traffic Light clicked' event block followed by a 'set pedxing to 1' block.</p>

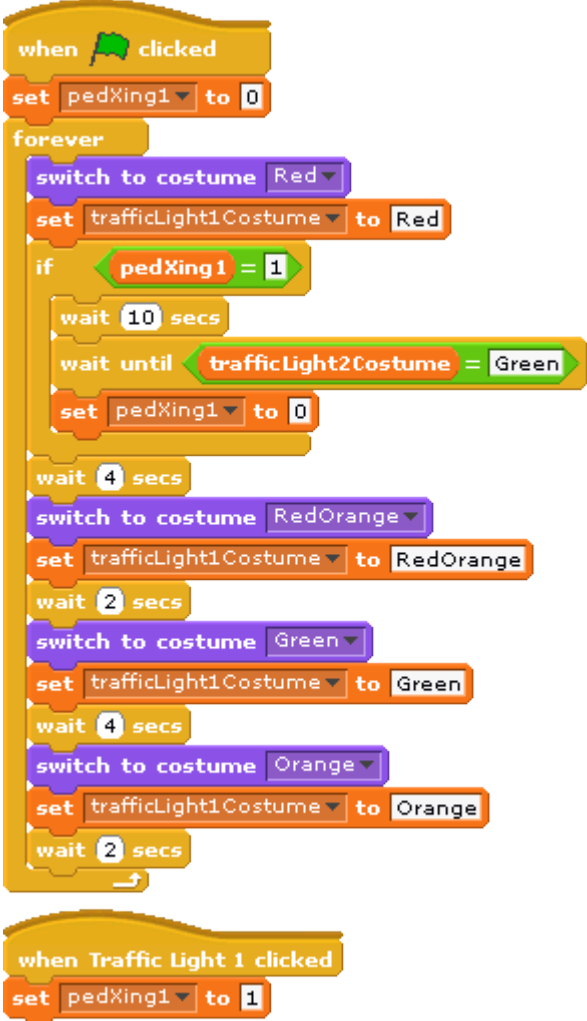
### Challenge 4

Have 2 traffic lights with their sequences linked like in challenge 1 and have pedestrian crossings on both. Bear in mind that stopping one traffic light delays its sequence from restarting and this will make the traffic lights' sequences out of sync. Therefore it would be difficult to get them back to normal but essential if both of them being green would cause a crash.

This task is especially difficult so teams may be unable to complete it and also they probably won't have enough time to attempt it. However any attempts made should still be useful by getting students thinking about how it could be done.

One possible approach is to first add pedestrian crossings like in challenge 3 but have different variables for each pedestrian crossing status (pedXing1 and pedXing2) to allow the pedestrian crossings to be independent. Next add variables to record each sprites current costume which is updated each time it changes (trafficLight1Costume and trafficLight2Costume). Then to get everything back to the right sequence in the 'if' statement which detects a pedestrian requesting to cross add a 'wait until' block below the 'wait 10 secs' block and check for the costume of the other light to be green then it will only continue when the lights are back in sequence (when one is red the other is green).

Here are the blocks used:

Traffic Light Sprite 1	Traffic Lights Sprite 2
 <pre> when clicked   set pedXing1 to 0   forever     switch to costume Red     set trafficLight1Costume to Red     if pedXing1 = 1       wait 10 secs       wait until trafficLight2Costume = Green       set pedXing1 to 0     wait 4 secs     switch to costume RedOrange     set trafficLight1Costume to RedOrange     wait 2 secs     switch to costume Green     set trafficLight1Costume to Green     wait 4 secs     switch to costume Orange     set trafficLight1Costume to Orange     wait 2 secs   </pre> <p>when Traffic Light 1 clicked set pedXing1 to 1</p>	 <pre> when clicked   set pedXing2 to 0   forever     switch to costume Green     set trafficLight2Costume to Green     wait 4 secs     switch to costume Orange     set trafficLight2Costume to Orange     wait 2 secs     switch to costume Red     set trafficLight2Costume to Red     if pedXing2 = 1       wait 10 secs       wait until trafficLight1Costume = Green       set pedXing2 to 0     wait 4 secs     switch to costume RedOrange     set trafficLight2Costume to RedOrange     wait 2 secs   </pre> <p>when Traffic Light 2 clicked set pedXing2 to 1</p>

The reason for storing values of the sprites costumes in variables is because it appears each sprite is unaware of other sprites. Therefore sprites cannot discover values about other sprites such as what is the other sprites current costume. However variables can be set to be available to all sprites (this is the default option) so work like global variables.

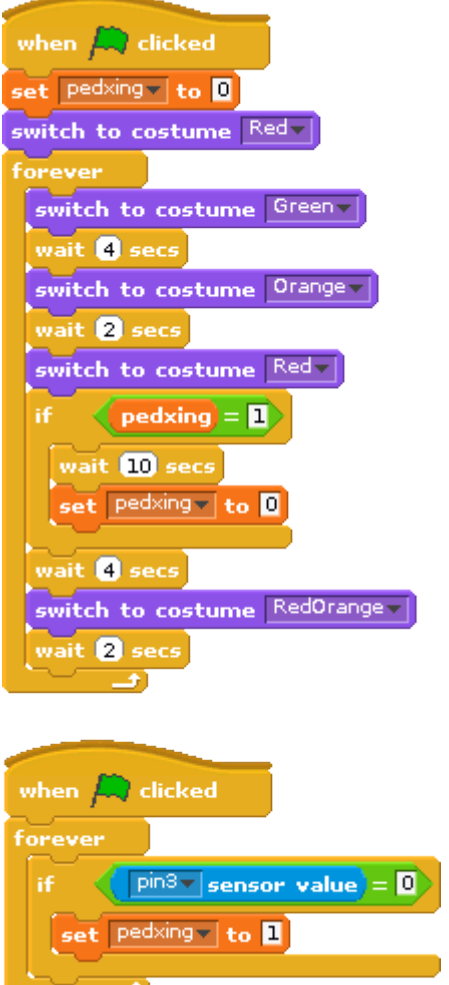
## Part 2

Note these tasks must be done using Scratch GPIO version 2 (ScratchGPIO2) not the regular Scratch, you will find a shortcut to ScratchGPIO2 on the desktop. This runs a program to handle the GPIO communications within Scratch, loads Scratch and enables Scratch's Remote Sensor Connections (RSC). This allows us to make use of the GPIO pins of a Raspberry Pi. When you load ScratchGPIO2 it opens a template file which is just a new file with RSCs enabled. However as it isn't the original new template it can easily be overwritten so it is important to do a save as when saving new projects for the first time otherwise it saves onto the default template for ScratchGPIO2.

### Task 1

If pins are not listed in the 'sensor value' block's drop down menu/list this probably means the Remote Sensor Connections are not enabled or didn't initialise properly. To solve this right click on a 'sensor value' block and choose "enable remote sensor connections"; if however it is already enabled then try disabling it and re-enabling it.

### Challenge 1

Details	Blocks
<p>Add a pedestrian crossing to 1 set of traffic lights like in part 1 challenge 3 (copy it if you wish) and add a paper clip switch like in part 2 task 1 to trigger the pedestrian crossing. It could perhaps use a Jelly Baby as the switch "The Jelly Baby arrives at the crossing".</p> <p>Connect jumper wires with paper clips in the end of them to pins 3 and 6 like in part 2 task 1.</p> <p>Next take the blocks from part 1 challenge 3 (you can copy its file to save time) and then add the checking of the pin 3 sensor value like in part 2 task 1 and when it is 0 (true) set pedxing to 1 to initiate the pedestrian crossing delay when the light is red.</p>	

## Challenge 2

Put a LED into or on the head of a Jelly Baby and then create an animation so that when the Jelly Baby has a thought it is shown in a thought bubble on the screen and the LED turns on and then when the thought has ended the light goes off.

1. Delete the cat sprite.
2. Add a Jelly Baby sprite - in the new sprite buttons click the 'choose new sprite from file' button and locate the Jelly Baby sprite in the things folder (if you can't see this folder you may not be in the costumes folder so click on the costumes button on the left).
3. Connect up a LED using pins 3 and 6 as shown in the instructions for part 2 task 2.
4. Add a 'when green flag clicked' block.
5. Add a broadcast to turn pin 3 off in case it has been left on.
6. Add a 1 second delay (this seems to solve a problem where the sequence becomes out of sync for a moment when it starts).
7. Add a forever loop with contents of: a broadcast to turn pin3on (turn the LED on), a think block (display a thought), a broadcast to turn pin3off (turn the LED off) and a wait for 2 secs (wait before stating again).



Times and messages can be adjusted if you wish.

## Appendix 12 – Case Study: Guidance for the event organiser

### Setting up the Raspberry Pi computers

#### *Adding sprites*

There are 2 sprites that were created for this tutorial which need adding to sub folders of Scratch's costumes folder. The costumes folder is usually located at:  
/usr/share/scratch/Media/Costumes

- The Jelly Baby sprite needs to go in the things folder
- The Traffic Light sprite needs to go in the transportation folder

Note: You may find you do not have rights to add to this folder and to solve this you need root privileges. You can either select "Open Current Folder as Root" from the tools menu when viewing the folder in the file manager which opens up a window of the folder with root privileges or you can copy the file with sudo from within the terminal/command line.

#### *Scratch GPIO version 2*

Part 2 requires Scratch GPIO version 2 to be installed (ScratchGPIO2) on the Raspberry Pi computers – see <http://cymplecy.wordpress.com/2013/04/22/scratch-gpio-version-2-introduction-for-beginners> for more details on it and how to install it.

#### *Python*

As Scratch GPIO version 2 uses Python with the Raspberry Pi GPIO libraries behind the scenes these must be installed. They are installed by default on Raspbian Linux for Raspberry Pi and should also be installed on other Linux distributions for Raspberry Pi; if however they are not installed you will need to install them.

#### **Introductions to each part**

It would probably be beneficial to provide introductions before each part begins and cover areas the students may struggle with or may not have previous experience with. Content covered depends on the skills of students in the group; typical areas to cover would be general introductions to the activities including:

#### *Part 1*

- Introduction to the Scratch interface
- Introduction to blocks
- Introduction to loops, conditional statements and setting conditions such as use of the equals operator, and perhaps variables
- Exporting and importing sprites

#### *Part 2*

- A basic electronics introduction – Ohms, circuits, purpose of a resistor (to reduce current), importance of wiring an LED correctly (it will light the wrong way round but won't use the diode which stops current from going the wrong way and thus protects the LED), etc.
- Introductions to using GPIO ports and connecting jumper wires, LEDs with resistors and so forth, perhaps with drawings on a board.
- Perhaps explain how in ScratchGPIO2 you must use save as when you first save a project so that you don't overwrite the default template. This is explained in the worksheets so could possibly skip this instruction if you feel the students will understand it from the worksheets.

- Explain circuits they will be creating, perhaps drawing them on a board.

### Task and challenge worksheets

When printing out the worksheets it is advisable to use the Arial font in size 12 or higher to make the content clear for students to read which is especially useful for students with learning difficulties such as dyslexia.

Keep the challenge worksheets separate with challenges per part on separate sheets. Then provide the students with challenges for the related part once they have finished the tasks for that part.

### Resources

You will require sets of the following items per team or per person depending on whether students will be working as teams or individuals:

- A Raspberry Pi
- Accessories for the Raspberry Pi – Keyboard, Mouse, Monitor, Power Supply, SD Card with an operating system on it (Raspbian is recommended)
- An adapter to connect the Raspberry Pi to a monitor
- Speakers or Headphones
- 3x LEDs (1x Red, 1x Orange or Yellow, 1x Green)
- 3x 220 ohm Resistors
- 6x Jumper Wires (female to female)
- 2x Paper Clips (small, thin and non-plastic-coated)
- 1x Jelly Baby (students may require more if they attempt the challenges which use Jelly Babies).

Having spare electronics and Jelly Babies is recommended.

Students will also require:

- Worksheets for the tasks
- Getting started with Scratch guides (optional dependant on their pre-existing Scratch skills)
- Challenges worksheets (provided once they have completed the tasks for the current part)

Event staff (those helping with the event) will require:

- Worksheets for the tasks
- Challenges worksheets
- Possibly getting started with Scratch guides if they haven't used it much before
- Advice for event staff document which includes task troubleshooting and solutions to challenges
- Details on what they should observe
- Event timetable
- Location details (directions, where to park, how to find rooms the event will be in or a meeting point, and so forth).

It is advised to distribute these prior to the event so the event staff are fully aware of what the event contains and what they will be required to do.

## Appendix 13 – Case Study: Survey

A survey will be conducted before and after the event to see if the event enhanced students' perceptions and understanding of computing and met the aims and objectives. Most questions will have 1 - 5 point answers in a likert scale. For these questions radio buttons for each value will be used if completed online or tick boxes if paper-based. Some questions will use text boxes for more flexible answers. Some other questions will have options to choose from as their answers and use radio buttons or check boxes<sup>143</sup> depending on if multiple options are applicable.

### Introduction for the surveys

The following will be added at the beginning of each survey.

This survey is to obtain details of secondary school students' opinions and understanding of computing as part of a dissertation project by Paul Albinson.

If you have any queries or wish to find out more about the research you can contact Paul by emailing [palbinson@bournemouth.ac.uk](mailto:palbinson@bournemouth.ac.uk). If a teacher is around while you are completing the survey they may be able to answer basic questions about completing the survey.

All questions require an answer unless they say optional beside them<sup>144</sup>.

Any information used will be published anonymously (it will be kept secret) and will not contain any information which could be used to identify you. By completing this survey you accept that the anonymous data you provide can be published. Participation in the survey is optional.

### Before event survey

#### *Students*

- What school do you go to/attend? (optional)
- Write 3 words that describe your opinion of computing<sup>145</sup>.

**Out of 5 where 1 is very unlikely and 5 is definitely (1 = very unlikely, 2 = unlikely, 3 = possibly, 4 = very likely, 5 = definitely)**

- How likely are you to choose ICT or Computing as a GCSE option or an equivalent?

---

<sup>143</sup> Boxes to tick will be used if paper-based survey is used.

<sup>144</sup> This may need updating on the different surveys as for example the online survey uses asterisks to show which questions are mandatory.

<sup>145</sup> This will be a text box for the students to write anything in.



- How likely are you to study AS/A level Computing or a college computing course?
- How likely are you to choose to study a computing course at university?
- How likely are you to apply for a course at Bournemouth University?
- How likely do you think you will get a job in the computing industry?

**Out of 5 where 1 is very unconfident and 5 is very confident (1= very unconfident, 2 = unconfident, 3 = slightly unconfident/anxious, 4 = confident, 5 = very confident)**

- If I asked you to describe an 'if' statement how confident would you be with your reply?
- If I asked you to describe what a loop is how confident would you be with your reply?
- If I asked you to describe what a variable is how confident would you be with your reply?
- How confident do you feel about using Scratch to program?
- How confident do you feel about learning new programming languages?
- How confident do you feel about using any programming language?

**In comparison to other students in your age group, out of 5 where 1 is poor and 5 is excellent (1 = poor, 2 = below average, 3 = average, 4 = above average, 5 = excellent)**

- How would you rate your computing skills?
- How would you rate your programming skills?

#### **Other questions**

- Does any member of your family work with computers? Yes/No  
If so, please select from the following sectors: Clerical/Administration/Secretary, Software or Web Development, Game Development, Media Production, Hardware or Networking, Military, Teaching Computing, Not sure/prefer not to say, other (please specify)
- Have you considered working in the computing industry after leaving education?  
Yes/No  
If so:
  - What part of the computing industry are you most interested in working in: Software or Web Development, Game Development, Media Production, Hardware or Networking, Teaching Computing, not sure, other (please specify)
  - What is your main motivation/reason for wanting to work in the computing industry? I think the computing industry would be interesting and rewarding, I

think I am likely to get a good job, I think I would earn a lot of money, I think working with computers is cool, other (please specify).

- Have you heard of anyone famous who works with computers?

If so, do they (please select): inspire you, mean nothing to you, bore you?

### **After event survey**

In addition to repeating the previous survey questions which relate to students' perceptions and understanding of computing to see if they were improved by the event the following questions will also be asked.

### **Teachers and students**

**Out of 5 where 1 is poor and 5 is excellent (1 = poor, 2 = below average, 3 = average, 4 = above average, 5 = excellent)**

- How would you rate the event overall?
- How would you rate the tasks?

### **Students**

**Out of 5 where 1 is extremely boring and uninteresting and 5 is very interesting and very enjoyable (1 = extremely boring and uninteresting, 2 = boring and uninteresting, 3 = tolerable/okay, 4= interesting and enjoyable, 5 = very interesting and very enjoyable).**

- How enjoyable and interesting was the day overall?
- How enjoyable and interesting was doing the basic tasks and animations in Scratch (part 1)?
- How enjoyable and interesting was using electronics (lights and switches) with Scratch (part 2)?

### **Teachers**

Teachers will also be asked to provide feedback on the event and specifically to cover the following points:

- Has the event been useful to you?
- Has the event increased your computing knowledge?
- Has the event provided you with ideas on interactive teaching methods?
- Has the event improved your understanding of physical computing and how it can be used with teaching computing?
- Has the event provided you with ideas for events you could run with the students (perhaps continuing on from the events activities or repeating them with new students)?

## Appendix 14 – Case Study: Observations

In addition to the formal surveys the observers (those helping run the event) will be asked to provide feedback on the effectiveness of the day with the following questions:

- In general did the students understand the tasks? Did any particularly struggle with the tasks?
- In general did the students seem motivated and keen to complete tasks to move onto the later more complex tasks and challenges?
- Did any students complete the tasks and have time to move on to the challenges?

If so:

- Did they enjoy the challenges?
- Did they struggle with the challenges?
- Were there differences between the skills of the groups/teams?
- Did any students show initiative and try other features and components that weren't specifically mentioned to enhance a task or try something different?

These questions will not have set answers to choose from and the observers will be asked to take notes during the day to help answer these questions. Observations shall be unobtrusive to provide unbiased results and to avoid distressing the students.

## Appendix 15 - Research Information Sheet

### Introduction to the research

This research is part of a Master's Degree dissertation project by Paul Albinson. The dissertation is an investigation into the reasons behind a noticed lack of interest in computing and ways to make computing more appealing. This is especially relevant at the moment due to the changes being proposed for the National Curriculum that make computing more prominent and increase the computing content covered.

The research you will be involved in is a case study of a university outreach computing event for secondary schools designed to enhance students' perceptions and understanding of computing and to provide teachers with a Continuing Professional Development (CPD) opportunity to learn more about computing and provide ideas on interactive teaching methods. It will include the use of physical computing such as showing the hardware which makes computers work and using electronics with computers as inputs of a program and for outputs as a result of coding such as controlling lights, sensors, motors and so forth. It is designed to make programming more fun and engaging by showing the effects of programming over a physical object for example turning on a light, interacting with sensors etc. and how inputs such as switches can be used.

Students will be provided with worksheets of tasks to do using Scratch and a Raspberry Pi computer. The students will work in teams or individually to complete as many tasks as possible within the allocated time. The tasks increase in complexity and make use of skills and concepts learned. There will also be problem solving challenges which have less detail and allow for flexibility and experimentation when developing the solutions.

### Aims and objectives

The project is designed to meet the following aims and objectives.

#### **Aims:**

- To enhance students' perceptions and understanding of computing via an outreach computing event
- To provide teachers with a Continuing Professional Development (CPD) opportunity to learn more about computing and provide ideas on interactive teaching methods

#### **Objectives:**

- To provide fun and motivational programming examples which demonstrate fundamental programming concepts, physical computing and programming with electronics
- To show the relevance of computing via hands-on examples ideally with as many real-world and relevant examples as possible
- To provide teachers and students with a CPD opportunity to learn more about computing and to provide teachers with ideas for activities they can run with their students (perhaps continuing on from the events activities or repeating them with new students)
- To observe measured improvement in students perceptions and understanding of computing

### **How results/data will be collected**

Students will be asked to complete a survey before the event on their opinions of computing and their current computing skills.

During the event observations of the students taking part will be carried out by the observers (those helping run the event). They will observe the success of the event and how students respond to it such as did students enjoy the event, were the tasks too difficult and so forth.

After the event students will be asked to complete surveys with similar questions to the surveys conducted before the event. This is to see if the event improved perceptions and understanding of computing and in particular programming. They will also be asked for their opinions of the event.

Surveys will be either online surveys or paper-based and observations will be written notes on paper or word processed.

After the event school staff who attended the event will be asked for their opinion of its effectiveness.

### **How the results/data will be used**

The results will be analysed to see what insights they provide into students' perceptions and understanding of computing and whether the event met its aims and objectives. The results and analysis will be included in the dissertation project. If you wish to obtain a copy of the results or to be notified of how you can get a copy of the completed research please contact the researcher by emailing [palbinson@bournemouth.ac.uk](mailto:palbinson@bournemouth.ac.uk).

### **Your rights**

The researcher appreciates your involvement in this research and it is very useful for their dissertation project. However there is no requirement to take part in any of the research and you can choose to opt-out from any part of it.

All data collected from or about students will be kept anonymous (it is kept secret) and secure and any published data will not be usable for identifying students.

Permission will be gathered to publish relevant information from discussions with school staff and this will not contain any personal data or data which can identify students.

Under the Freedom of Information Act you have the right to access any information held or produced by public authorities. Therefore the dissertation will be publically available once completed. If you wish to be provided with any data collected or to be notified of how you can get a copy of the completed research please contact the researcher by emailing [palbinson@bournemouth.ac.uk](mailto:palbinson@bournemouth.ac.uk).

To comply with the Data Protection Act any data collected will be kept secure while it is in use and only agreed data will be published. Also data will be destroyed as soon as it is no longer relevant or required.

### **Contact**

If you have any queries, wish to find out more about the research, or if you have a complaint you can contact the researcher (Paul Albinson) by emailing [palbinson@bournemouth.ac.uk](mailto:palbinson@bournemouth.ac.uk).

## Appendix 16 – Suitability of the Raspberry Pi

Due to the complexity and difficulties found with using the electronic components on the Raspberry Pi its suitability could be questioned. This was discussed with Alastair who had previously considered creating a lot of content around using the Raspberry Pi but the problems encountered at the event have prompted him to reassess this. Also the practicalities of using a Raspberry Pi such as time to set it up each lesson<sup>146</sup> are another consideration. Although the Raspberry Pi is cheap it can get expensive when buying accessories for it (power supply, case, cables, keyboard, mouse etc.) and if the organisation wishes to set up the equipment permanently alongside computers to share desk space and monitors, keyboards and mice a KVM<sup>147</sup> switch is required further increasing costs and set up complexity. The usefulness of a Raspberry Pi computer needs to be assessed against the organisation's needs. Whereas it is good for physical computing due to its GPIO ports it may not justify the cost and complexity along with other limitations such as its small amount of memory and CPU speed. The use of GPIO ports for programming can be used directly with a computer via a USB to GPIO adapter or breakout board<sup>148</sup>. The use of electronics can be simplified further by using equipment such as a PicoBoard (The Playful Invention Company 2010) which is a board with sensors and controls which can interact with Scratch without needing to do any electronics. However if you wish to use the Raspberry Pi, which will allow for more complex custom electronics, you can simplify electronics and expand its capabilities (such as adding more inputs/outputs) by using an expansion board such as a PiFace (Element14 2013a) or a Gertboard<sup>149</sup>. Alternatively a breadboard, such as (Amazon, 2013), can be used to simplify connecting electronics and are very cheap (from around £3).

### References

- Amazon, 2013. *BB400 Solderless Plug-in BreadBoard, 400 tie-points, 4 power rails, 3.3 x 2.2 x 0.3in (84 x 55 x 9mm)*. Amazon. Available from: <http://www.amazon.co.uk/BB400-Solderless-Plug--BreadBoard-tie-points/dp/B0040Z1ERO> [Accessed 17 August 2013].
- Diolan, 2013. *USB-GPIO Interface Adapters Comparison*. Diolan. Available from: [http://www.diolan.com/io/digital\\_in.html](http://www.diolan.com/io/digital_in.html) [Accessed 17 August 2013].
- Element14, 2013a. *PiFace Digital for Raspberry Pi*. Element14. Available from: <http://www.element14.com/community/docs/DOC-52857> [Accessed 17 August 2013].
- Element14, 2013b. *Assembled Gertboard for Raspberry Pi*. Element14. Available from: <http://www.element14.com/community/docs/DOC-51726> [Accessed 17 August 2013].
- Numato Lab, 2013. *8 Channel USB GPIO Module with Analog Inputs*. Numato Lab. Available from: <http://numato.com/8-channel-usb-gpio-module> [Accessed 17 August 2013].
- Proto-PIC.co.uk, 2013. *Breakout Board for CP2103 USB to Serial w/ GPIOs*. Fife: RelChron Limited. Available from: <http://proto-pic.co.uk/breakout-board-for-cp2103-usb-to-serial-w-gpios> [Accessed 17 August 2013].

---

<sup>146</sup> Many institutions may not have the ability to permanently keep the Raspberry Pi computers set up due to for example the need to use the room with regular computers as well.

<sup>147</sup> Keyboard, Video and Mouse.

<sup>148</sup> See (Diolan 2013; Numato Lab 2013; Proto-PIC.co.uk, 2013) for a few examples and more information.

<sup>149</sup> See (Element14 2013; Raspberry Pi 2013) for more information.

Raspberry Pi, 2013. *Gertboard is here!* Cambridge: Raspberry Pi. Available from: <http://www.raspberrypi.org/archives/1734> [Accessed 17 August 2013].  
The Playful Invention Company, 2010. *PicoBoard*. The Playful Invention Company. Available from: <http://www.picocricket.com/picoboard.html> [Accessed 16 August 2013].

## Appendix 17 – Additional survey results

Here are results for the remaining survey questions which provide additional background to students’ perceptions and understanding of computing.

### Future ambitions: Studying at Bournemouth University

An additional future ambitions questions was asked to see if the students plan to study at Bournemouth University “BU” (their local university<sup>150</sup>) and if the event improved their opinion of the university<sup>151</sup>. The surveys’ results are in figures 1 and 2.

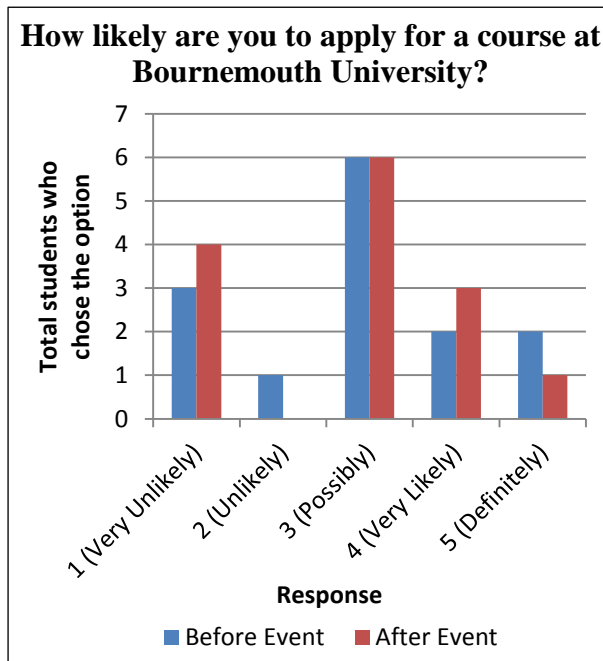


Figure 1: How likely students will apply for a course at Bournemouth University

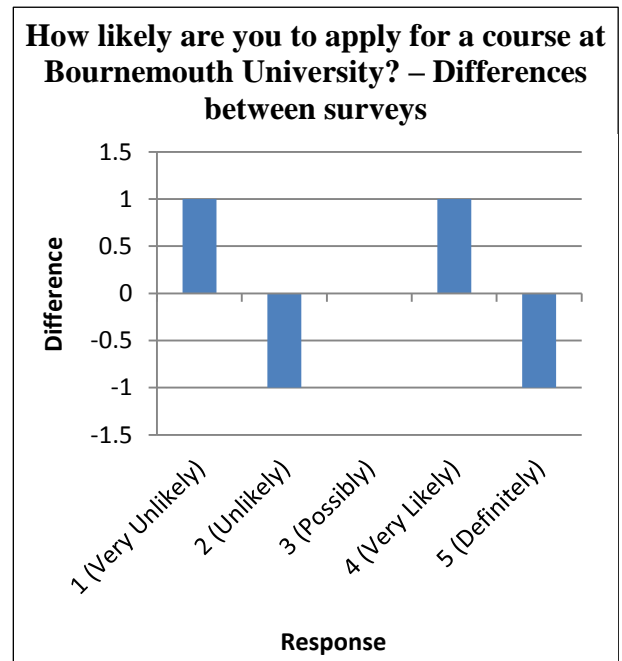


Figure 2: How likely students will apply for a course at Bournemouth University - Differences between surveys

The surveys’ results show the majority of students may ‘possibly’ (42.86%) apply for a course at BU; responses were 28.57% positive, 42.86% neutral and 28.57% negative which remains the same for both surveys despite some changes in responses. There was a slight reduction in interest after the event with one less ‘definitely’ response and one more ‘very likely’ response<sup>152</sup> and one more ‘very unlikely’ response and one less ‘unlikely’ response<sup>153</sup>.

<sup>150</sup> This is also the university supporting this research.

<sup>151</sup> It was hoped that the event would be enjoyable and as it was organised via BU it would be effective promotion for the university.

<sup>152</sup> This could be that the one less ‘definitely’ response became ‘very likely’ suggesting one student became a little less certain Bournemouth University is suitable for them.

<sup>153</sup> This could be that the one less ‘unlikely’ response became ‘very unlikely’ suggesting one student became a little less certain Bournemouth University is suitable for them.



Results are significantly less positive than those considering studying computing at university thus indicating these students are not sure if BU is the right university for them (it is a possibility not a certainty).

The averages and quartiles for the surveys (Tables 1 and 2, and Figure 3) show on average opinions of BU slightly reduced as a result of the event. The first quartile reduced by 0.75 resulting in a wider range of results and difference between low and high results. The mean average reduced by 0.14. The differences between surveys are not statistically significant ( $U=94$ ,  $Z=0.1608$ ,  $P=0.87288$ )<sup>154</sup>.

Table 1: Quartiles for the “How likely are you to apply for a course at Bournemouth University” question

	Before Event	After Event
<b>Minimum</b>	1	1
<b>Quartile 1</b>	2.25	1.5
<b>Median (Q2)</b>	3	3
<b>Quartile 3</b>	3.75	3.75
<b>Maximum</b>	5	5

Table 2: Averages for the “How likely are you to apply for a course at Bournemouth University” question

	Before Event	After Event	Difference
<b>Mean</b>	2.93	2.79	-0.14
<b>Median</b>	3	3	0
<b>Mode</b>	3	3	0

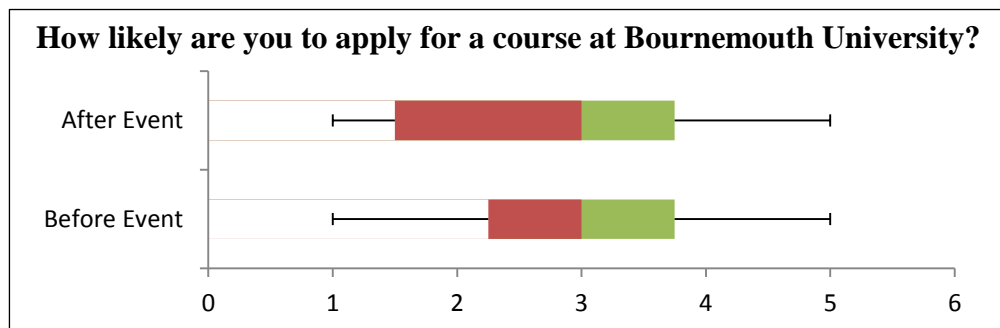


Figure 3: Box plot for the “How likely are you to apply for a course at Bournemouth University” question

<sup>154</sup> Coincidentally this is the same statistical significance scores as the “How likely students will choose to study a computing course at university” question. There probably isn’t a correlation between the results but it could be that the same amount of differences in responses happened with the before and after surveys for both questions.

### Other

The students were also asked to write at least 3 words that describe their opinion of computing; only 1 student answered this question and said “interesting, fun, relaxing”.

The students were also asked whether they had heard of anyone famous who worked with computers and what their impressions of them were<sup>155</sup>. Unfortunately only 28.57% answered these questions. Results for these questions are in figures 4 and 5 which show only 50% of respondents had heard of any famous computing people with responses evenly split between positive and negative impressions of them.

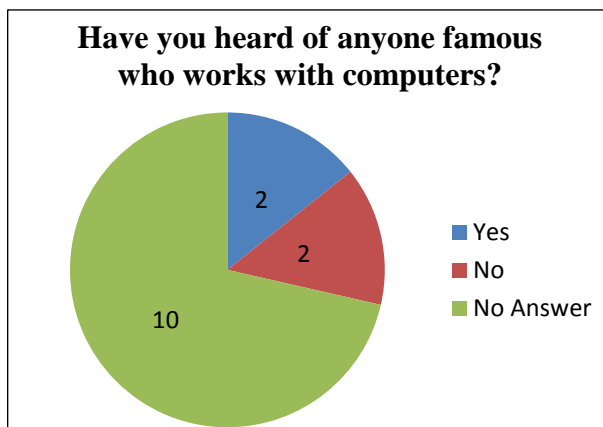


Figure 4: Students who have heard of any famous computing people

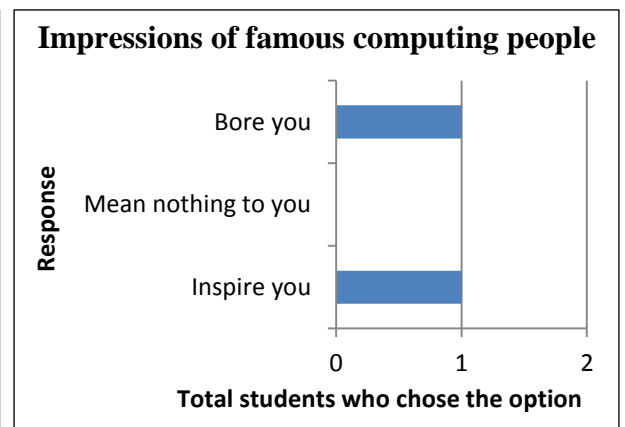


Figure 5: Students impressions of famous computing people

Students were also asked if any of their family members work with computers and if so what sector they work in<sup>156</sup>; responses are shown in figures 6 and 7. These questions were asked to see if decisions were influenced by computing careers being popular within their families. Out of the 78.57% of students who replied only 27.27% said family members work with computers. These family members' jobs are

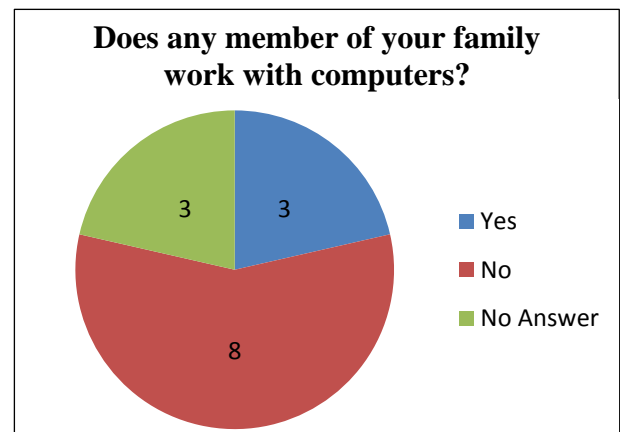


Figure 6: Students who have family members who work with computers

<sup>155</sup> These questions were only asked in the before the event survey as the event wouldn't change the results as it doesn't discuss famous computing people.

<sup>156</sup> These questions were only asked in the before event survey as the students only need to answer these questions once (the event won't change these facts).

a business on eBay and repair<sup>157</sup>. These responses show low amounts of family connections to the computing industry which may influence decisions.

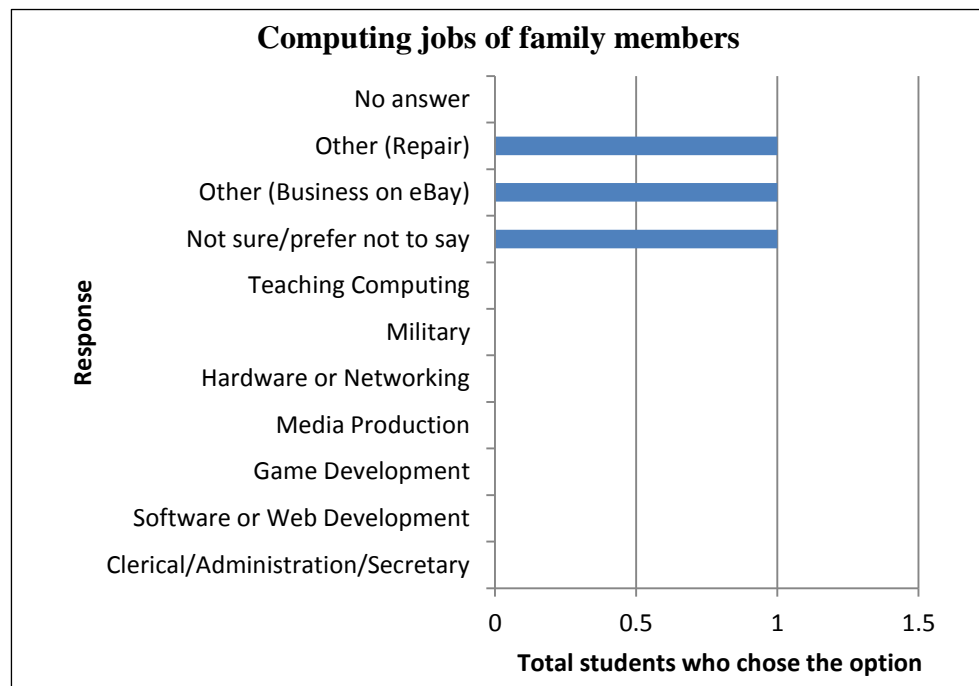


Figure 7: Computing jobs of family members

## Discussion

### Studying at Bournemouth University

The majority of students said studying at Bournemouth University was a possibility (42.86%) or ‘very likely’ or ‘definitely’, the positive responses, (28.57%) but there were some reductions in positivity after the event.

### Other

- 50% of respondents had heard about famous computing people with responses evenly split between positive and negative impressions of them.
- 27.27% of respondents have family members who work with computers with jobs listed as a business on eBay and repairs.

<sup>157</sup> One student chose “Not sure/prefer not to say”

## Appendix 18 – Learning resources and links

Here is a list of learning resources and links you may find useful.

### Professional bodies, working groups and government organisations

- BCS, The Chartered Institute for IT (<http://bcs.org>) – The professional body for IT
- The BCS Academy of Computing (<http://academy.bcs.org/>) – A BCS academy to promote, support and enhance computing education
- Computing At School Working Group “CAS” (<http://computingatschool.org.uk/>) – A grass roots community organisation which promotes computing at schools and supports education providers to enhance computing teaching.
- Network of Teaching Excellence in Computer Science (<http://www.computingatschool.org.uk/index.php?id=noe>) – A collaboration between CAS and BCS to provide teachers via CPD opportunities around enhancing computing education
- Department for Education (<http://www.gov.uk/df>) – The department of the UK government responsible for education and children’s services

### Computing clubs

- Code Club (<http://codeclub.org.uk/>) – After school computing clubs for 9-11 year olds
- Technocamps (<http://www.technocamps.com/>) – Free computing workshops for 11-19 year olds
- Coding for kids (<http://codingforkids.org>) – A community to support teaching programming and computational thinking to children
- Young Rewired State (<https://youngrewiredstate.org/>) – A network of designers and developers aged 18 and under. It is part of rewired state and is designed to encourage children to code.

### Online learning

- Khan Academy (<https://www.khanacademy.org/>)
- Coursera (<https://www.coursera.org/>)
- Code Academy (<http://www.codecademy.com>)
- FutureLearn (<http://futurelearn.com/>)
- Udacity (<https://www.udacity.com/>)
- EdX (<https://www.edx.org/>)

- Harvard open courses at Harvard Extension School  
(<http://www.extension.harvard.edu/open-learning-initiative>)
- Cambridge GCSE Computing online (<http://www.cambridgegcsecomputing.org/>)

### Teaching resources

- CS unplugged (<http://csunplugged.org/>) - Highly recommended free practical learning activities for teaching computing concepts without the need for computers. The activities are ideal for any age especially young students.
- Apps for Good (<http://www.appsforgood.org/>) – Trains educators to support students with creating applications via an applications course and mentoring
- Computing ITT (<https://sites.google.com/site/primaryictitt/>) – An extensive list of teaching resources available for primary schools
- Computer Science For Fun “CS4FN” (<http://www.cs4fn.org/>) – Fun interactive computer science resources
- Computer Science teaching resources from The Royal Society of Edinburgh  
([http://www.royalsoced.org.uk/1034\\_ComputingScience.html](http://www.royalsoced.org.uk/1034_ComputingScience.html))
- GCSE Computing for Schools books (<http://www.lulu.com/spotlight/susanjrobson>)
- Recommendations from CAS members - Note: Membership of CAS is required to access these resources.
  - Resource sets (<http://community.computingschool.org.uk/set>) – Resources added by CAS members and grouped into sets by themes such as qualification, programming language and so forth
  - Recommended books for teaching computing  
(<http://community.computingschool.org.uk/forums/63/topics/1021>)
  - A list of recommended computing books  
(<http://community.computingschool.org.uk/resources/199>)
  - Suggestions for teaching basic logic/algorithms to years 4-6  
(<http://community.computingschool.org.uk/forums/3/topics/677>)
  - Textbook recommendations for A Level Computing  
(<http://community.computingschool.org.uk/forums/23/topics/1204>)
  - OCR GCSE Computing: An Unofficial Teacher's Guide  
(<http://community.computingschool.org.uk/resources/378>) – A guide for teaching OCR GCSE Computing

- Computational thinking resources  
(<http://community.computingatschool.org.uk/resources/252>)
- CSc resources (<http://cscresources.wordpress.com/2013/04/16/links-to-programming-resources/>) - Links to programming resources
- Teach ICT (<http://teach-ict.com/>) – ICT and Computing teaching resources
- KS2 Skills Progression in Scratch (<http://code-it.co.uk/year4/scratchprogression.htm>)
- An AppInventor workshop (<http://www.hannahdee.eu/appinventor/>) – This is a complete workshop in a box and has everything you need to run it (handouts, slides, notes and so forth).
- Bubble Sorting Algorithm Demo (<http://theingots.org/bubblesort/>)
- OCR Raspberry Pi (<http://www.ocr.org.uk/qualifications/by-subject/computing/raspberry-pi/>) – Resources for teaching with the Raspberry Pi
- Feynlabs – Using the Raspberry Pi to teach Computer Science  
(<http://www.opengardensblog.futuretext.com/archives/2013/02/feynlabs-using-the-raspberry-pi-to-teach-computer-science.html>)
- Yousrc (<http://www.yousrc.com/>) – A free web-based environment for teaching programming

#### Robots to teaching computing

- Roamer <http://www.roamer-robot.com/public/> & <http://www.valiant-technology.com/uk/pages/corphome.php>
- Bee-bot <http://www.tts-group.co.uk/shops/tts/Range/Bee-Bot/92b201eb-0c85-4e38-a297-35932cbc56b6> & [http://www.kenttrustweb.org.uk/kentict/kentict\\_ct\\_bee.cfm](http://www.kenttrustweb.org.uk/kentict/kentict_ct_bee.cfm)
- Pixie <http://www.swallow.co.uk/pixie/pixie1.htm>
- Pixie versus Bee-Bot [http://www.kenttrustweb.org.uk/kentict/kentict\\_ct\\_bee\\_pix.cfm](http://www.kenttrustweb.org.uk/kentict/kentict_ct_bee_pix.cfm)

#### Other

- Computing++ (<http://www.computingplusplus.org>) – A scheme to provide support from industry to schools
- Code.org (<http://www.code.org>) – A non-profit foundation which supports computer programming education