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[1] 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 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

[1] 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.

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[2]. 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[3] undergraduate majors[4]. 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.

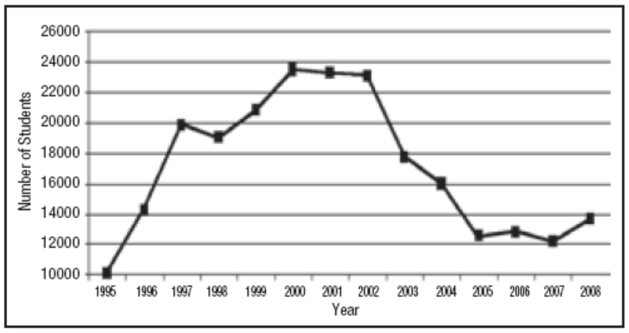

Figure 1 – Taulbee Survey: Newly Declared CS/CE Undergraduate Majors (Zweben 2009)

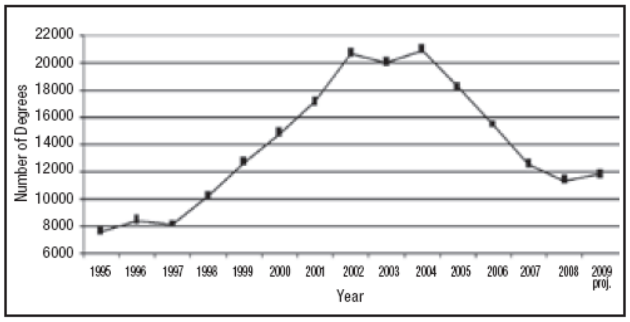

Figure 2 – Taulbee Survey: CS & CE Bachelor's degree production (Zweben 2009)

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.

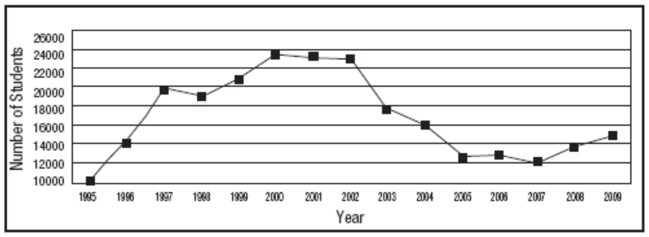

Figure 3 – Taulbee Survey: Newly Declared CS/CE Undergraduate Majors (Zweben 2010)

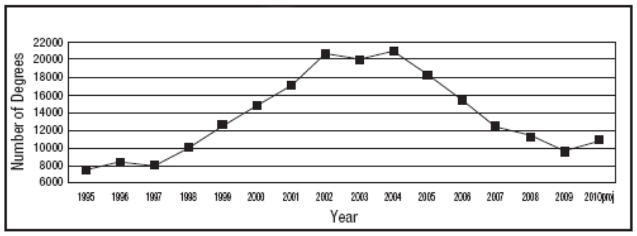

Figure 4 – Taulbee Survey: CS & CE Bachelor's degree production (Zweben 2010)

[2] http://www.cra.org/resources/taulbee/

[3] Computing Engineering

[4] There was however a slight decrease in 1998.

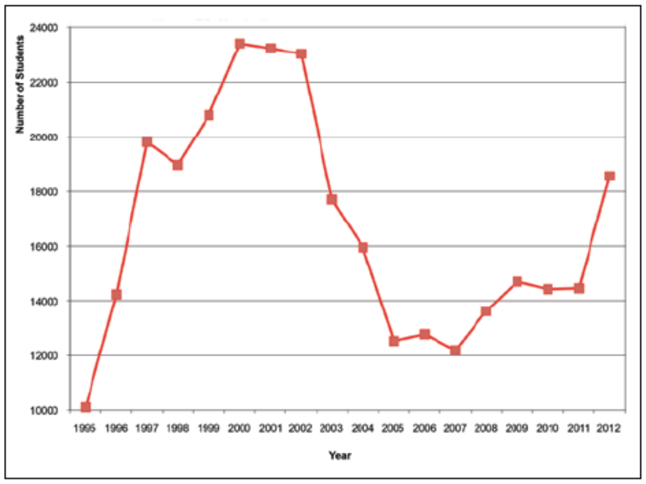

Figure 5 – Taulbee Survey: Newly Declared CS/CE Undergraduate Majors (Zweben 2013)

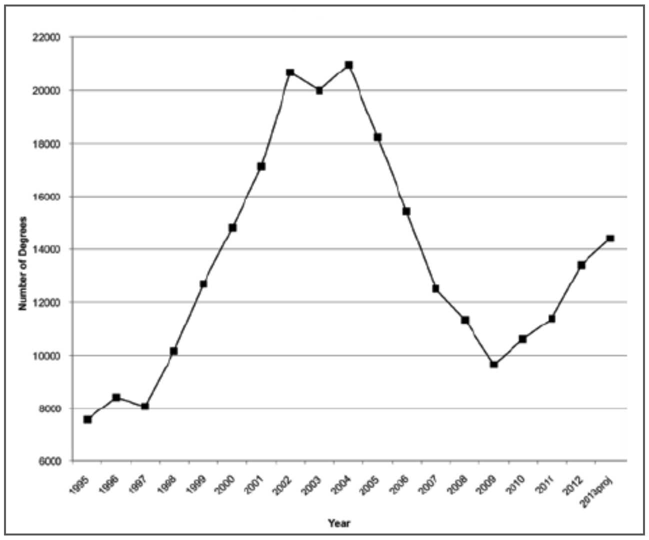

Figure 6 – Taulbee Survey: CS & CE Bachelor's degree production (Zweben 2013)

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.

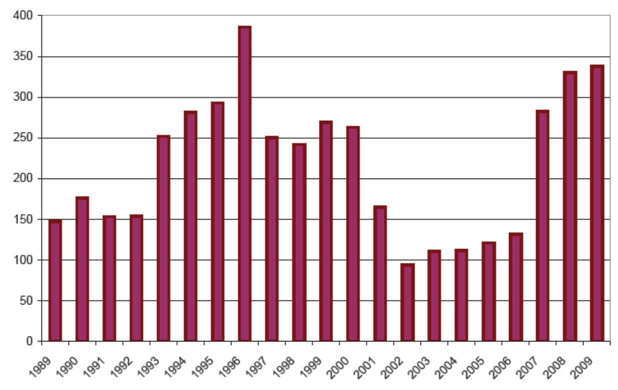

Figure 7 – Enrolments in Harvard's CS50 course (Malan 2010)

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[5] 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 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[6], 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)[7]. 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[8] 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 say how students' consider their ICT fluency skills as good (faculty

[5] Traditionally programming tutorials begin with how to create a program to write "Hello World" to the screen.

[6] Students are typically only taught how to use the Microsoft Office suite of office applications and similar basic computer uses such as web browsing.

[7] Naace, ITTE, and the Computing at School Working Group (2012) provide a more in-depth comparison of ICT and Computer Science.

[8] This is the entire UK school system from ages 5-16.

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[9]. 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 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[10], Code.org[11], Google Computer Science for High School[12] and Google Summer of Code[13]. In addition there are plenty of other resources such as online courses like Coursera[14] and Khan Academy[15] 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 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

[9] It was seen as impractical to base an entire course on the ITF framework.

[10] http://www.codeclub.org.uk/

[11] http://www.code.org/

[12] http://www.cs4hs.com/

[13] https://developers.google.com/open-source/soc/

[14] https://www.coursera.org/

[15] https://www.khanacademy.org/

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[16], App Inventor[17] and Alice[18]. 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

[16] http://scratch.mit.edu
[17] http://appinventor.mit.edu
[18] http://www.alice.org

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 is 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 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 [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 [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 [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/MehranSahami.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 [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/crndocs/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/crndocs/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/crndocs/issues/0513.pdf [Accessed 17 May 2013].