Visualising the code: a study of student engagement with programming in a distance learning context

Elaine Thomas, Soraya Kouadri Mostéfaoui, Helen Jefferis

School of Computing and Communications, The Open University, UK, elaine.thomas@open.ac.uk, soroya.kouadri@open.ac.uk, h.jefferis@open.ac.uk.

Abstract

Programming is a subject that many students find difficult and it may be particularly challenging for distance learning students working largely on their own. Many ideas have been put forward in the literature to explain why students struggle with programming, including: the relative unfamiliarity of computer programming or 'radical novelty' (Dijkstra, 1989), cognitive load (Shaffer, 2004) and that the whole learning environment may be influential (Scott & Ghinea, 2013).

This paper reports on the first phase of a project, 'Visualising the code', which is investigating the impact of using a visual programming language on student engagement with programming. We used as our case-study, TU100 'My digital life' which is a level 1 undergraduate Computer Science module in the Open University (UK). The rationale for this work stems from the necessity of developing an introductory undergraduate module that will engage students of widely differing prior levels of experience in terms of programming and of education generally. In TU100, the module team introduced a visual programming environment, based on Scratch (MIT, 2007), called 'Sense' which is used in conjunction with an electronic device, the SenseBoard.

We analysed the grades of 6,159 students in the final assessment across six presentations of the module to identify student performance in the programming task, as distinct from their overall performance on the module. The aim was to explore whether there was any difference between student engagement with the programming task in comparison with non-programming tasks. Early results suggest that there is no significant difference in levels of engagement between these tasks, and it appears that success, or otherwise, in one type of task is a good predictor of engagement with the other task.

There are implications for networked learning of this work, given that the learning environment encompasses: the student's own home or other space, both printed books and digital learning materials, a programming environment linked to a physical device (i.e. the SenseBoard) and communications networks that link students to their peers and to their tutors. The learning environment also includes support through face-to-face and online tutorials and other online resources, such as forums.

In the next phase of the project we will analyse the textual comments made by TU100 students in the end of module survey to evaluate their views on the visual programming environment.

Keywords

visual programming, Scratch, student engagement, distance learning, computer science education, networked learning.

Introduction

In recent years, the teaching of Computer Science has moved up the national agenda in the UK as the government has required fundamental changes to the way that computer programming is taught in schools (Gove, 2012). At the heart of the new curriculum lie skills in algorithmic thinking and programming which are now compulsory from the age of six until sixteen. At higher education (HE) level, it is widely acknowledged that students find programming difficult and drop-out rates from Computer Science courses are high. Various theories have been put forward to explain this difficulty – some of these are discussed in the next section. Learning to program can be particularly challenging for distance learning students because there is no support immediately to hand. Programming skills are considered to be essential for a Computing and IT degree so an important issue for the academics involved in developing a new level 1 distance learning module was, how to

ISBN 978-1-86220-337-2

introduce new students to programming in a way that will engage them but not alienate more experienced students.

This paper reports on the work in progress of a project, 'Visualising the code', which is investigating the impact of using a visual programming environment on student engagement with programming. We are using as our case study, a large-scale 60 credit level 1 distance learning Computing and Information Technology module, TU100 'My digital life'. The project also investigates, what impact, if any, does not completing the programming questions have on students' engagement with the non-programming elements of the assessment within the module. For example, some students may excel at programming but not in other aspects of a broad-based Computing and Information Technology module and vice-versa.

We will first present a brief overview of the literature around the teaching and learning of programming to explain some of the difficulties that students experience, then we set the context of the research project followed by our research methodology and analysis of the results. In the process, we make some tentative links between theories about programming education and theories about networked learning.

Introducing students to programming

There is considerable debate about effective methods of introducing programming to students at higher education level. Mostly, the studies are concerned with the teaching of programming in a traditional university setting. Dropout rates for first year students in computing subjects are high across the HE sector in the UK (Lee, 2017) and many students find learning to program difficult (Jenkins, 2002). There are different interpretations as to the reasons for this, perhaps because different skills and types of learning are involved. Programming demands a high abstraction level and generalised way of thinking. Programming also requires a good level of both knowledge and practical problem-solving techniques; as well as practical and intensive study skills. Dijkstra (1989) contends that learning to program entails 'radical novelty' as novices may not have acquired the necessary prior skills on which to build and progress. Jenkins (2002) argues that the reason behind the difficulties in learning to program is the blend of learning types required: surface learning for remembering features such as syntax and order of precedence, and deep learning in the understanding of concepts and development of true competence. Shaffer et al (2004) suggest that utilising cognitive load theory can aid the teaching of novice programming and Computer Science students.

Scott and Ghinea (2013) focus on the whole learning environment explaining that 'soft scaffolding', detailed feedback and motivational practices are most likely to be effective. Possibly, networked learning, as characterised by the integration of information and communication technologies in conjunction with the furthering of connections between students, between students and tutors and between the learning community and learning resources (Dirckinck-Holmfeld, 2016) may provide a perspective for examining programming education.

Other research suggests that student engagement is an important issue. Gomes and Mendes (2007) detail a list of factors that need to be considered when teaching programming in order to achieve better student engagement. Finally, it requires using innovative ways of teaching due to its dynamic nature. Eltegani and Butgereit (2015) identify attributes that result in students not engaging with programming including: the style of education doesn't appeal to all students; students may be overwhelmed by what they have to do, educators may not be sensitive to students' responses to educational methods, amongst others.

The visual programming environment

Scratch is a visual programming environment developed by the MIT (2007). Instead of typing in text, students 'drag and drop' blocks of code which snap into position like jigsaw pieces. All the usual programming constructs, such as statements, conditions, loops, operators, variables, and so on, are provided as blocks. The idea is that students can quickly learn to produce programs without having to worry about syntax such as spelling conventions and punctuation, of a particular language. In this way 'cognitive load' (Sweller, 1988) is reduced as students do not have to remember the syntax and can focus on building the program. One approach to improving learning is to provide multiple sensory input (Shaffer et al, 2004). The act of dragging pieces of code together rather than typing in code is physical one, especially as the blocks of code 'snap' together when they are arranged in the correct way.

ISBN 978-1-86220-337-2

Although this environment was developed for school-age students, a number of studies carried out with entry-level university students in the United States suggest that Scratch is effective as an introductory programming language. For example, Malan and Leitner (2007) used Scratch at Harvard Summer School for two weeks as a precursor to Java. In an end of course survey, 76% of the students surveyed indicated that Scratch had a positive effect on them, 8% rated the effect as negative while 16% indicated that learning Scratch had no effect on their performance in the Java programming course. Rizvi et al (2011) used Scratch with computer science students who were struggling with programming in a semester-long course. They reported that an improvement in performance and better retention amongst their target group students. A study of a two-week course for new engineering students in India (Mishra et al, 2014) reported mixed results. On one hand, the performance of novice students was on a par with more experienced students in learning and applying programming concepts in Scratch. However, novice students lagged behind their more experienced peers when they progressed to the textual language C++.

Context of the project

Unlike other universities, the OU (UK) does not require any previous educational attainment as an entry condition. Therefore, TU100 assumes only a basic level of prior computer use; the module is a level 1 undergraduate module worth 60 credits and is part of a common stage 1 curriculum which is taken by all Computing and IT students. It covers a broad range of computing and networking topics with an emphasis on ubiquitous networked computing. Programming activities are woven through this material; however, they are only one aspect of the overall study.

Previous teaching of programming at level one in a 30 credit module used a text-based programming environment, JavaScript, but over half of students avoided answering the question on programming in the End of Module Assignment (EMA). Feedback from students and tutors suggested that JavaScript was unpopular with students (Richards & Woodthorpe, 2009). Therefore, when TU100 was developed the module team[1] decided to use a visual programming environment to teach programming. While a visual programming environment has limitations in terms of employability, it allows students to learn fundamental skills and concepts and build their confidence to enable them to succeed with other programming languages. Therefore, Sense (Open University, 2011), a version of Scratch, was specially developed for the module.

Alongside the programming environment, students use the SenseBoard, a small programmable hardware device that can be connected to the student's computer using a USB connection. The SenseBoard contains input capabilities such as a slider device and a button along with sensors, such as a microphone, a thermistor and motion detector which allowed a program to respond to changes, for example so that an on-screen graphic could be controlled by a slider. The board also includes outputs such as LED lights. The SenseBoard was intended to be mini laboratory that could represent the workings of sensors and actuators in the real world (Thomas et al, 2014).

TU100 is designed to be delivered by distance learning and is presented in a combination of printed and online self-study materials including resources such as audio and video recordings. TU100 has two presentations per year, each spread over nine months, with the October presentation followed by a February one. Although numbers of students on the module are large, the student experience is very different from a MOOC (Massive Open Online Course). At the OU(UK) students have a named tutor who supports them by offering face-to-face and/or online tutorials using an audio-conferencing system and also moderates online forums. Although distance learning tends to be a rather solitary activity, online asynchronous communication tools, such as forums, blogs and wikis are used for communication between students and their peers, their tutors and the module team. This can be seen as a networked learning environment in the sense described by Goodyear et al (2014) as being concerned with connections between learning activity and the physical places, and tools that include digital, material and hybrid artefacts.

Engagement with course materials in general and programming in particular presents challenges for distance learning students as they may have little recent experience of formal education and they may be working full or part-time whilst studying, and some may have caring responsibilities. Therefore, there were concerns amongst faculty staff that students who struggled with the programming could avoid the programming assessment but still pass the module by simply working harder on the other module topics.

[1] In the OU (UK) courses are developed by module teams.

ISBN 978-1-86220-337-2

The research

The first stage of the 'Visualising the code' project focussed on student engagement with programming in the End of Module Assignment (EMA). Student progress on TU100 is assessed by a series of interactive Computer Marked Assignments (iCMAs), five Tutor Marked Assignments (TMAs) and a final, larger EMA. The latter is a scenario based piece of work that ties together and examines many of the overall themes and learning outcomes. For example, one year the context involved a fictitious Non-Government Organisation (NGO) recruiting a new employee to provide a bridge between trustees, staff and supporters of the NGO and the technical staff on a pilot project in rural Nepal. The assessment consisted of five tasks that were being used as part of the second stage of the interview process. The tasks included: writing a short report on the digital divide which should discuss how data is stored, the implications of the Data Protection Act, calculations of data transfer rates, and the use of solar panels.

In the programming task students were asked to create a prototype system for monitoring mudslides during monsoons. This involved using the SenseBoard's microphone and slider to simulate the noise and movement associated with a mudslide and getting the board's LEDs to flash as warning lights. An example of a completed part of this program is shown in Figure 1, note the use of the sensors from the SenseBoard.

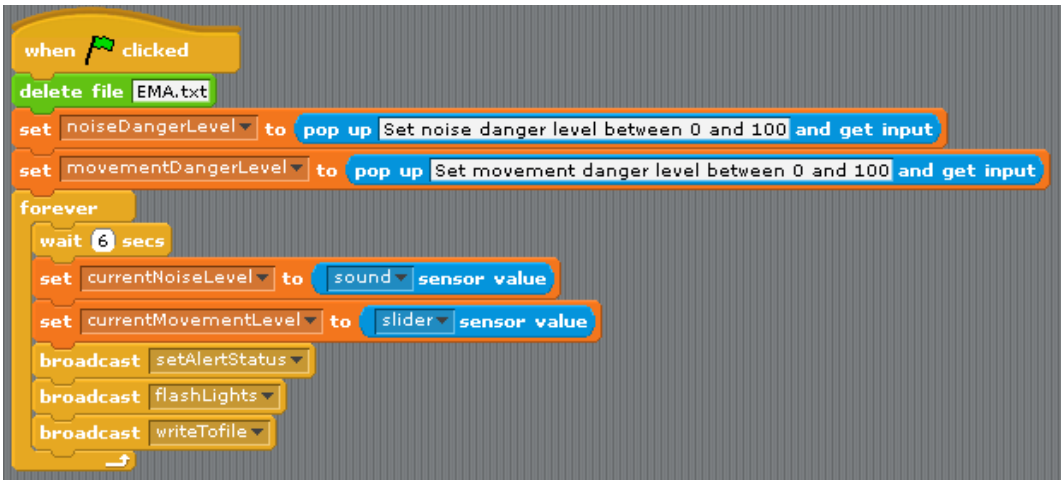

Figure 1 Example of completed programming assessment task

Methodology

Quantitative methods were chosen for this stage of the project to analyse student performance across the whole EMA, their engagement with the programming task and their scores for this task in comparison with their achievements in the remaining, non-programming Computer Science tasks. Our data collection consisted of two phases: firstly, identifying the programming question in each EMA for a total of six presentations of TU100 and, secondly, obtaining data on student performance in the EMA from the University. The resulting data was then analysed using the SPSS Statistics Package V22.

Students' EMA scores were collected for six separate presentations, starting with October in 2013 and ending with February 2016. As the EMA covers material from the whole module, the programming elements were identified to create split scores for each student showing their programming and 'other' marks. These values were then standardised as the percentage of marks awarded overall for programming varied for some years (for the first 2 presentations programming contributed 16.8% whereas for later years it contributed 20% to the overall final mark).

Results and discussion

In total, the EMA results for 6159 students from all the six presentations were analysed to investigate trends in terms of student engagement with programming as compared with the non-programming elements of the modules. Firstly, we looked at students' scores for each element by cohort and compared the cohorts over the six presentations. Next, we looked at individual students' performances and, finally, we looked the scores of students who had failed the module.

ISBN 978-1-86220-337-2

Figure 2 Comparison of mean scores, programming vs non-programming, for the six presentations

The graph in Figure 2 shows a comparison between the mean scores for the programming, the non-programming elements and the average result overall for each of the six student cohorts. The red line shows the number of students (right hand axis) on each presentation.

In October 13/February 14 the average scores for the programming and non-programming elements were about the same and in October 15/February 16 this was also true, although students seem to have done slightly better in the programming elements. In October 14/February 15, students have an average score for the programming element which is around 15% higher than for the non-programming element. This suggests that possibly the programming question was easier in these EMAs. The mean and median for the overall score, the programming task score and the non-programming tasks score along with the standard deviation for the 6159 students from the six presentations are shown in Table 1.

Table 1 Mean, median and standard deviation of the overall scores for the programming and non-programming tasks

	Mean	Median	Standard. Deviation
Overall score	71.3	76.0	19.1
Programming score	76.7	88.1	29.2
Non- programming score	70.0	74.5	19.1

To further investigate the relationship between these two elements, we produced a scatter graph of each student's results comparing their programming score with their non-programming score.

The spread of results makes it difficult to see a clear picture. However, looking at the detail, it is possible to interpret the results of the analysis. For example, there is a cluster of student scores in the top right-hand corner, these are the scores of the students who did well in both the programming and the other elements of the assessment. In the bottom right-hand corner, there are the scores of a few students who performed well in

ISBN 978-1-86220-337-2

programming, but not well in the other areas. In the opposite top left corner, are the scores of students who did badly on programming, but scored well in the other tasks. It is also notable that a small number of students scored zero on programming, this is shown by the left-hand vertical line.

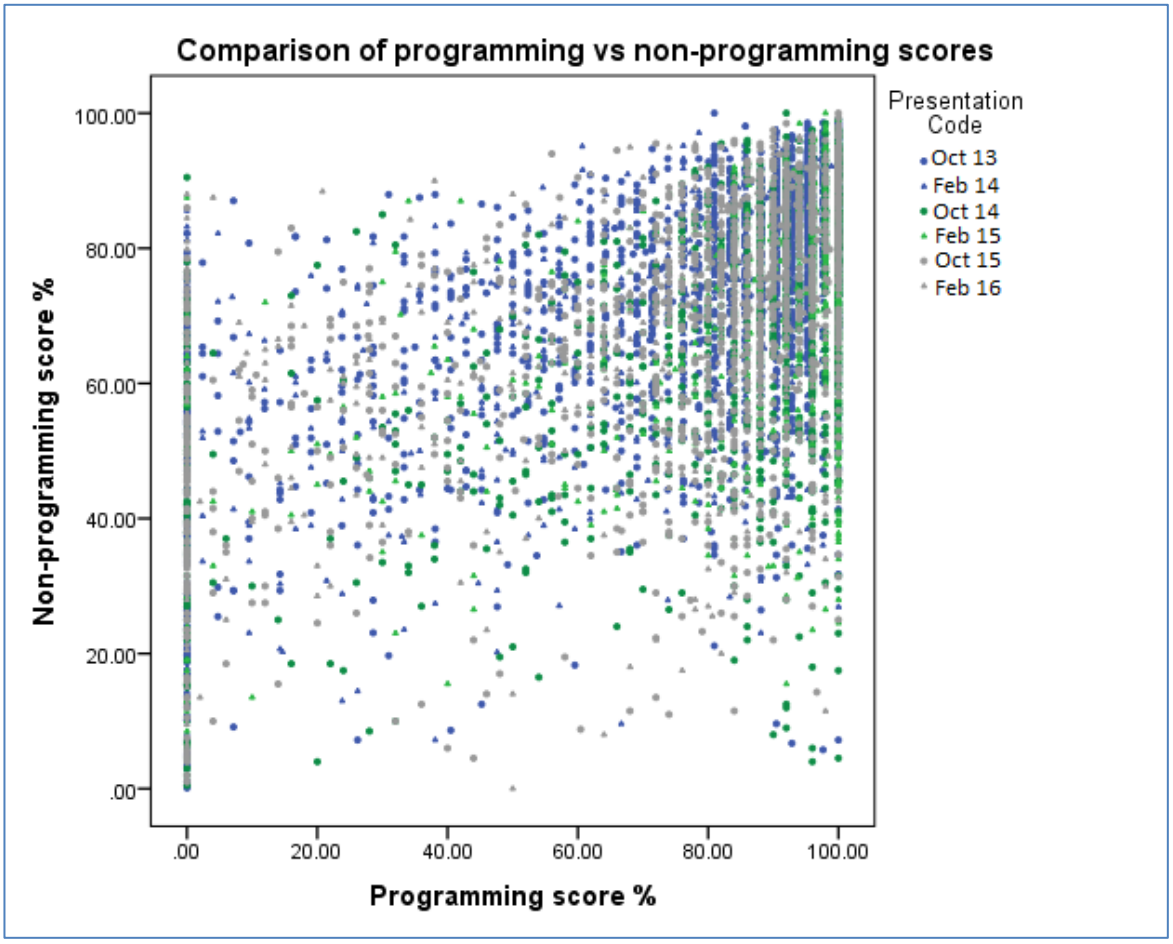

Figure 3 Comparison of programming and non-programming scores for the six presentations

Overall there is a strong ($r=-0.543$, $p<0.01$) correlation between the scores that students achieved in the programming and non-programming elements of the final assessment, i.e. students who did well in programming also tended to do well in the non-programming elements, and vice versa. This does not vary much by presentation as shown in Table 2 below.

Table 2 Correlation between scores for programming and non-programming elements

Presentation	Correlation (In all cases p <0.01)
October 2013	.569
February 2014	.602
October 2014	.522
February 2015	.512
October 2015	.551

ISBN 978-1-86220-337-2

February 2016	.555

Next, we looked at the scores of students who failed the programming element of the final assessment (i.e. those who scored less than 40% - the pass mark for TU100) to see if many were still managing to pass overall. A total of 816 students out of the total of n = 6,159 were in this group, but of these only 460 of them managed to score more than 40% overall and so pass the module. This represents less than 7.5% of students, which shows that very few students who failed to engage with programming were able to reach the pass grade. This is a much higher level of engagement than in the predecessor module where only around 50% of students chose a programming task in the final assessment.

Our findings from the combined results of n=6159 students over six presentations of the module suggest that there is a strong correlation between performance in the programming element and the non-programming elements of the level 1 undergraduate distance learning Computing and IT module.

The multi-modal nature of learning when using a visual programming language (Shaffer et al, 2004) may be particularly engaging for TU100 students because the Sense programming environment is combined with the physical SenseBoard device. Cognitive load theory (Sweller, 1988) may be influential in that struggling with syntax is particularly difficult and frustrating for novice programmers in a distance learning context and the Sense visual programming environment avoids this problem. Also ‘soft scaffolding’ support (Scott & Ghinea, 2013) is provided to TU100 students in the form of practical programming activities, feedback on the tutor-marked assignments along with the use of synchronous and asynchronous communication tools that provide tutor support and promote peer-to-peer support. In this sense, the networked learning aspects of the learning environment may be the determining factor in promoting student engagement with programming. The interaction between the student working on programming activities in their own place and their own time, but linked through digital communications networks to their tutors and their peers, and using physical and digital resources represents a ‘hybrid space’ as described by Ryberg et al (2016). This interaction may be the important factor that enables students to engage successfully with the programming element.

This study has its limitations in that we do not know whether the programming task had any influence on the students who did not submit the final assessment and so failed the module. Also, more detailed work is needed to investigate how the small number of students who did not engage with the programming task but passed TU100 fare in later, more advanced, programming modules. However, the results from this project suggest, that using a visual programming environment as part of a broader Computing and IT course can be very successful in encouraging students to engage with programming. This in itself improves students’ confidence so there may be less ‘radical novelty’ (Dijkstra, 1989) concerning programming.

Conclusions and future work

In this study, we looked at how level 1 Open University students performed in their assessment after they had been taught how to program with a visual programming environment (Sense). We also investigated whether students' performance in programming was significantly different from their performance with the non-programming tasks in the final assessment (EMA). A statistical analysis of the students' EMA scores over six different presentations suggests that overall there was not a significant difference in performance between the two elements, neither for the cohort as a whole nor for the individuals, but rather that success, or otherwise. in one is a good predictor of performance in the other. Very few students (7.5%) passed the module without engaging with the programming element of the assessment, which is a considerable improvement on the predecessor module which used JavaScript. This suggests that using a visual programming environment has been successful in engaging students in the programming tasks.

In our future work, we intend to explore students’ views about learning with the visual programming environment by carrying out a qualitative analysis of textual comments provided by students in end of module surveys. Additionally, a similar visual programming environment is now being used in a new OU(UK) level 1 module to introduce students to programming. This will be followed by another level 1 module using a text-based language (Python). We intend to monitor students' performance for both the programming and non-programming elements of the new modules and conduct a similar quantitative and qualitative analysis.

References

Dirckinck-Holmfeld, L. (2016) Networked learning and problem and project based learning – how they complement each other. In: S. Cranmer, N. Dohn, M. de Laat, T. Ryberg, J. A. Sime (Eds) Proceedings of the 10th International Conference on Networked Learning. Lancaster University, Lancaster, UK, pp. 200-220. http://www.networkedlearningconference.org.uk/past/nlc2016/abstracts/pdf/P14.pdf (accessed 3rd October 2017).

Dijkstra, E.W. (1989) A Debate on Teaching Computer Science: On the Cruelty of Really Teaching Computer Science. Communications of the ACM, 32, 12, 1398-1404. http://www.cs.utexas.edu/users/EWD/transcriptions/EWD10xx/EWD1036.html (accessed 19th July 2017).

Eltegani, N. & Butgereit, L. (2015) Attributes of students engagement in fundamental programming learning International Conference on Computing, Control, Networking, Electronics and Embedded Systems Engineering (ICCNEEE), September 2015, pp.101-106.

Gomes, A; Mendes, A. J. (2007), An environment to improve programming education. In Proceedings of the 2007 ACM International conference on Computer systems and technologies, New York, USA, 2007 88:1-6.

Goodyear, P., Carvalho, L., & Dohn, N. B. (2014). Design for networked learning: framing relations between participants activities and the physical setting. In S. Bayne, C. Jones, M. de Laat, T. Ryberg, & C. Sinclair (Eds.), Proceedings of the Ninth International Networked Learning Conference, Edinburgh, UK, pp. 137–144 http://www.lancaster.ac.uk/fss/organisations/netlc/past/nlc2014/abstracts/pdf/goodyear.pdf (accessed 3rd October 2017).

Gove, M. (2012) Digital Literacy and the Future of ICT in Schools. Presentation at the BETT Show, Department for Education, 'Digital literacy campaign – Michael Gove's speech in full' The Guardian, 11th January 2012, https://www.theguardian.com/education/2012/jan/11/digital-literacy-michael-gove-speech (accessed 26th September 2017).

Jenkins, T. (2002) On the Difficulty of Learning to Program. In Proceedings of the 3rd Annual HEA Conference for the ICS Learning and Teaching Support Network, 1-8. (accessed 19th July 2017).

Lee, G. (2017) Which universities have the highest first year dropout rates? Channel 4 FactCheck 28th August 2017. https://www.channel4.com/news/factcheck/which-universities-have-the-highest-first-year-dropout-rates (accessed 19th January 2018).

Malan, D. J and Leitner, H. H. (2007) Scratch for Budding Computer Scientists, SIGSCE'07, Covington, KY, 223-227.

Mishra, S. Balan, S., Iyer, S. & Murthy, S. (2014) Effect of a 2-week scratch intervention in CS1 on learners with varying prior knowledge. In Proceedings of the 2014 conference on Innovation & technology in computer science education (ITiCSE '14). ACM, New York, NY, USA, 45-50.

MIT (2007) Scratch [Computer program]. Available at https://scratch.mit.edu (accessed 30th September 2017).

Richards, M. & Woodthorpe, J. (2009). Introducing TU100 'My Digital Life': Ubiquitous computing in a distance learning environment. In Ubicomp 2009, 30 Sep - 3 Oct 2009, Orlando, Florida, USA. http://oro.open.ac.uk/26821/2/ubicomp_final.pdf (accessed 19th July 2017).

Rizvi, M. Humphries, T., Major, D., Jones, M. & Lauzun, H. (2011) A CS20 course using Scratch. Journal of Computing in Small Colleges, 26:3 pp 19–27.

Ryberg, T., Davidsen, J. & Hodgson, V. E. (2016) Problem and project based learning in hybrid spaces: nomads and artisans. In: S. Cranmer, N. Dohn, M. de Laat, T. Ryberg, J. A. Sime (Eds) Proceedings of the 10th International Conference on Networked Learning Lancaster University, UK: Lancaster, pp. 200-220. http://www.networkedlearningconference.org.uk/past/nlc2016/abstracts/pdf/P14.pdf (accessed 3rd October 2017).

Scott, M. A. & Ghinea, G (2013) Educating programmers: A reflection on barriers to deliberate practice https://www.heacademy.ac.uk/knowledge-hub/educating-programmers-reflection-barriers-deliberate-practice (accessed 10th July 2017).

Open University (2011) Sense [Computer program]. Available at http://sense.open.ac.uk (accessed 30th September 2017).

Shaffer, D., Doube, W. & Tuovinen, J. (2003) Applying Cognitive Load Theory to Computer Science Education. In M. Petrie & D. Budgen (Eds) Proceedings of the Joint Conference of the European Association of Science Editors (EASE) & the Psychology of Programming Interest Group (PIGG), Keele, UK (accessed 26th September 2017).

Sweller, J. (1988) Cognitive load during problem-solving: Effects on learning. Cognitive Science 12 (2) pp257-285.

Thomas, E., Walker, S. & Richardson, P. (2012). What did the Romans ever do for us? 'Next generation' networks and hybrid learning resources. In V. Hodgson, C. Jones, M. de Laat, D. McConnell, T. Ryberg T & P. Sloep (Eds) Proceedings of the 8th International Conference on Networked Learning, 02-04 April, 2012, Maastricht, The Netherlands pp.337-344.

Acknowledgements: this work has been partially funded by eSTEeM the OU(UK) centre for STEM pedagogy http://www.open.ac.uk/about/teaching-and-learning/esteem/.

ISBN 978-1-86220-337-2