Does Scratch improve self-efficacy and performance when learning to program in C#? An empirical study.

Keith Quille
keith.quille@nuim.ie
Department of Computer Science
Maynooth University
Co Kildare, Ireland

Susan Bergin[S, C]
susan.bergin@nuim.ie
Department of Computer Science
Maynooth University
Co Kildare, Ireland

S = Senior Author, C = Corresponding Author

Abstract

This paper describes a study conducted in the 2015-2016 academic year, while examining three previous years of data. This paper investigated when students learnt Scratch, a block type programming language, at the same time as their object orientated introductory programming module which was delivered using C# (CS1), would their programming self-efficacy increase (a prominent factor in student success), and therefore as a result, their performance in the CS1 module. The second language was selected, specifically on the grounds that it may increase student programming self-efficacy. Scratch was chosen as students do not need to learn code syntax, rather it is a programming by discovery language. Arguably it may help struggling novice programmers to comprehend coding concepts (even threshold concepts) that they have not grasped in their mainstream text based language. This may be due to the fact that in a text based programming language such as Java or C#, if a student could not master the syntax, they may not be able to continue to cover more advanced concepts and may fail or be forced to drop out. This paper discusses both quantitative and qualitative findings in detail. The quantitative data consists of students programming self-efficacy and end of year results, collected at the beginning and the end of the academic year respectively while the qualitative data consists of a student survey administered at the end of the academic year. This paper concludes with recommendations and future work that would be valuable to educators considering similar pedagogical approaches to improve performance.

Keywords

Scratch, CS1, Introductory Programming.

Introduction

Computer Science (CS) is widely acknowledged for having a higher than average attrition rate, coupled with lower than average pass rates compared to other higher level domains. In Ireland this is no exception, where CS non-progression rates are highlighted year on year, by the Higher Education Authority (HEA). In two reports by the HEA (2010 and 2016 respectively) CS non-progression rates were reported at 25%, which is significantly larger than the 16% national average (Liston, Frawley, & Patterson, 2016; O. Mooney, Patterson, O'Connor, & Chantler, 2010). At honours degree level, CS has the highest non-progression of all domains. In addition, CS was one of only two domains where progression rates reduced from 2010 to 2016. The other domain was construction, where the non-progression rates can be attributed to the housing

collapse in Ireland during this era. Despite these concerns employment in CS is growing, with employment growth double that of the national GDP. Currently there are over 4500 unfilled vacancies in Ireland (Keenan, 2013), with this trend continuing into 2016 (O'Brien, 2016) with an expected 825,000 unfilled vacancies in ICT in Europe by 2020 (McGuire, 2015). This deficit in demand is echoed across the world, where the United States of America is set to have a shortfall of 1,000,000 developers and engineers by 2020 (www.code.org, n.d.).

It is well acknowledged both anecdotally and in a large body of literature that a significant contributor to this phenomenon, is the difficulty that students have in mastering fundamental programming concepts. Thus any novel pedagogical approaches to improve pass rates and reduce attrition are valuable to the CS education (CSEd) community. Some approaches / research documented in the literature include but are not limited to: using online video tutorials to facilitate student learning of threshold concepts (Hegarty-kelly, Lockwood, Bergin, & Mooney, 2015), teaching programming via Scratch to improve problem solving skills (Kalelioğlu & Gülbahar, 2014), integrating Problem Based Learning into existing CS courses (Kelly et al., 2004), facilitating student learning in computer science: large class sizes and interventions(Nolan, Mooney, & Bergin, 2015), using neurofeedback to promote student engagement(Lockwood, Mooney, & Bergin, 2016) and strategies for sharing lecture notes on VLEs (Mooney & Bergin, 2013).

2. Motivation and Related Literature

The authors of this paper are members of the Computer Science Education Research Group (CSER) at Maynooth University which has over 100 collective years of teaching experience. The authors between them have a large body of research on factors that influence student success and on predicting student performance in an introductory programming module (Bergin & Reilly, 2005, 2006, Quille & Bergin, 2015, 2016; Quille, Bergin, & Mooney, 2015). This research spans a decade and has produced some significant results, combined with a computational model and real time system which can predict student success at only four to six hours into the delivery of an introductory programming module. The accuracy of this model is ≈ 80% and has been validated on multiple data sets. More information can be found on this system and the validation studies in the references listed in this section.

A prominent factor in the prediction model, is a student's programming self-efficacy. This factor had a positive relationship with programming success. Other literature has also

found programming self-efficacy to be a predictor of success or in some way related to attrition (Campbell, Horton, & Craig, 2016; Vivian, Falkner, & Falkner, 2013; Watson, Li, & Godwin, 2014; Wiedenbeck, 2005). The research group is constantly examining methodologies to increase student programming self-efficacy. There is little literature in the space of the use of Scratch in third level, the small amount that exists, is usually part of a pre-third level course, summer course or CS0 module and not used or examined in the initial academic year (Armoni, Meerbaum-Salant, & Ben-Ari, 2015; Malan & Leitner, 2007). However one study attempted to introduce CS1 via Scratch (De Kereki, 2008). The study while finding no improvements in results found an increase in student motivation. Stajkovic reports that self-efficacy makes an important contribution to work motivation (Stajkovic, 2016), thus improvements in motivation due to the inclusion of Scratch, may be the result of increases in self-efficacy. Kereki only introduced Scratch for the initial three weeks of a fifteen week CS1 module, then reverted to Java. A longer period of Scratch use, may have value. If Scratch can therefore increase student self-efficacy, it may ultimately lead to increased success or results. Thus two hypotheses are formed:

H1: Scratch delivered in parallel to CS1, increases student programming self-efficacy.

H2: Scratch delivered in parallel to CS1, increases student performance.

The authors proposed the delivery of a Scratch module in parallel to the staple introductory CS1 programming module. Programming in Scratch consists of dragging and snapping blocks of code together to construct a program (MIT media lab, 2005). The blocks are predefined and available through a sorted visual display thus allowing a programmer with no previous knowledge to explore and find and code block required. This form of interaction alleviates syntax and compile errors or the requirement to know the code before a programmer may begin to build. Scratch can be used directly in a web browser thus avoiding the installation of an IDE (Integrated Development Environment) and in the case of Java, the JVM (Java Virtual Machine) which can be a complex task in itself for a novice programmer.

Scratch may allow novice programmers to reach threshold concepts before they are reached in the CS1 module or allow the students to access the threshold concept, even if they have not mastered the required CS1 code. A threshold concept is a concept that is likely to trouble a student as it is a previously inaccessible way of thinking about something (Meyer & Land, 2003). In computer science, in particular programming, the initial two threshold concepts that are reached are conditional statements and iteration, which are both regarded as challenging

concepts in introductory programming modules (Cherenkova, Zingaro, & Petersen, 2014). Thus giving more exposure to the threshold concept even if the student has not mastered the code may prove beneficial. This additional Scratch module should not take from the hours allocated to the traditional CS1 module. Recent research has reported positive student feedback while using Scratch in an introductory programming module which correlates with the theory that Scratch may aid students in understanding threshold concepts such as loops, with an example: "Though we did not learn Java syntax by using Scratch, we learned the type of thinking necessary to implement simple programs.... I was able to approach the first Java programs with an idea of how to tackle the problems. Though I did not yet know how to create a for-loop, I knew when a for-loop was necessary because I had used loops in my Scratch program" (Armoni et al., 2015).

Finally, as both modules conclude there will be two forms of reporting. A quantitative approach which will analyse the results of the CS1 module and compare the results to previous years where Scratch was not delivered and a qualitative approach which will survey students to examine their experiences of the inclusion of Scratch and benefits felt, if any.

3. Background

This study was conducted at an Institute of Further Education over four years, from 2012-2015 inclusive. The Institution delivers an Advanced Certificate in Software Development (ACSD) course which is at Level 6 on the National Framework of Qualifications (NFQ). The introductory programming module that was taught would arguably be comparable to many introductory programming modules delivered at Degree level. This introductory programming module was identical throughout the four years of this study. Entry to the course is based solely on an interview, thus students were not selected competitively based on grades. The introductory programming module was taught as one of three programming modules. The other two modules were Event Driven Programming (including introduction to SQL and databases) and Object Orientated Programming. The modules were delivered in consecutive block format, with the introductory programming module taught first, from September to November. The point value of the course and breakdown of content is discussed further in Section 4. It should also be noted that the overall ACSD course consisted of ten modules: Introductory Programming module (CS1), Event Driven Programming (EDP) using Graphical user interfaces (GUI), Object Orientated Programming (OOP), Games Development, Software

Architecture, Mathematics, Web Development, Project Management, Work Experience and Communications. Ethical approval was granted for the study, and the data was collected via an online system (Quille et al., 2015).

Table 1 highlights some specific cohort differences between each year group. This is to illustrate the cohort contrast over the four years and to show that no additional biasing was introduced, as all students in each year participated in this study, irrespective of grade or any other defining attribute.

Attribute	2012-2013	2013-2014	2014-2015	2015-2016
Number of Students	24	31	28	30
Male to Female Ratio	21:3	28:3	24:4	29:1
Mature to Non-Mature Ratio	4:20	13:18	9:19	2:28
Average Grade in CS1 X	70.58%	72.18%	69.53%	67.20%
Std. Dev. (grades in CS1) σ	16.81%	16.73%	18.00%	12.91%

Table 1: Contrast in cohorts over the four years

(A mature student is any student 23 years of age or older, a non-mature student is under the age of 23)

4. Methodology

This section details the two teaching and learning methodologies employed, in the introductory programming module and the Scratch module. This section concludes with the methodologies employed to investigate the two hypotheses formed in Section 2.

4.1 Introductory Programming module

The CS1 introductory programming module was compulsory for each student enrolled in the Advanced Certificate in Software Development course. The CS1 module was taught using C# and the .NET platform. Visual Studio 2010, 2012 and 2015 was used over the four years. In the initial three years of this study, no other programming was taught alongside the introductory programming module. The CS1 module produced a console application and covered the following topics: variables, printing, input, conditional statements, iteration, arrays and lists, (recursion and 2d arrays were briefly covered but were outside the scope of the course), scope, procedures, methods and functions. The module was taught over 10 weeks and consisted of the following structure: four hours of lectures per week and four hours of laboratories a week. The grade point score from the CS1 module was calculated from two continuous assessments and one written examination. The first continuous assessment (CA) consisted of a menu format, at

least one conditional statement and at least one loop and had an overall grade weighting of 30%. The second CA assignment consisted of similar topics as the first CA, but had additional complexity of procedures, methods and functions using additional complex data types such as an array. The overall grade weighting for the second CA was 40%. The final written examination was two hours in duration carrying an overall grade weighting of 30%.

4.2 Scratch – Game Development Module

The overall games development module was delivered as two separate independent modules. First Scratch was delivered as an introductory standalone games module in semester one alongside CS1. The module used the Scratch 2.0 IDE. The aims of this module was to introduce students to computer games, before they undertook the more advanced second element which consisted of Unity and C# scripting which was delivered in semester two. This introductory module ran for the same 10 weeks as the introductory programming module and consisted of one hour of lectures and one, hour and twenty-minute laboratory per week. The module structure consisted of: basic movement, costumes and costume animation using costume lists, conditional blocks, iteration, multi-threading movement concepts, variables, floor movements, (basic) gravity engine, projectile motion using quadratic equations, collisions and interactions, and finally a scoring system. The aim of this module was to conclude with the students building a complete 2D game, where there was a start screen, at least one level or an infinitely long game where scores are accumulated, and finally an end screen where the results or achievements are displayed. The grade point score for this module was calculated from one CA and one written examination. The CA accounted for 60% while the written examination accounted for the remaining 40%.

4.3 H1: Scratch delivered in parallel to CS1, increases student programming self-efficacy.

As part of the previous research conducted by the authors, programming self-efficacy was measured using a modified Rosenberg self-esteem questionnaire which consisted of 10 questions (Bergin, 2006). The programming self-efficacy data was captured using a web based version of the PreSS computational model (Bergin, 2006) named PreSS# (Quille et al., 2015). This study using the same methods as PreSS, collected student programming self-efficacy data when students were introduced to iteration, the second threshold concept. This may be where

the students programming self-efficacy could be at its lowest, therefore if the introduction of Scratch was to have any significant effect, it could be captured at this point.

4.4 H2: Scratch delivered in parallel to CS1, increases student performance

First the overall results in CS1 module were examined for the initial three years of delivery of the course. This was to investigate if the results from each year were significantly different. If this produced a result where there was no significant difference between the results of each of the three years, this would infer consistency before the addition of Scratch. Secondly the fourth year results were examined to see if any significant difference existed when compared to the initial three years results. This would then conclude if the delivery of Scratch alongside a CS1 introductory programming module significantly improved student performance.

5. Results

5.1 Quantitative Analysis

The section reports on two analyses, the self-efficacy data and the overall programming performance. In both cases, an analysis of variance was used (ANOVA). The ANOVA analysis measures the variance between multiple groups assuming independence of the variance in each group. The ANOVA analysis is similar to running multiple independent *t*-test's but is known to result in less type 1 errors, which relates to less false positives (Horn, 2008).

Student Self-Efficacy

Student self-efficacy data was collected from 2013 to the 2015. The collection system (PreSS#) was not complete during the initial study year of 2012. As per previous studies the 10 recorded self-efficacy values were fed into a data reduction algorithm, namely Principal Component Analysis (PCA). This method is outlined in detail in previous research (Bergin, 2006; Quille et al., 2015).

ANOVA Analysis **2013-2014, 2014-2015, 2015-2016**	
Total Students	88
Total Mean	-0.02906
Total Std. Dev.	1.36755
f-ratio	2.3822
p-value	0.0985

Table 2: ANOVA analysis of student's self-efficacy from the last three year groups

The results presented in Table 2, found no significant difference in the student programming self-efficacy between the three year groups, with a *p value* > 0.05.

Student Performance

An ANOVA analysis was conducted on the student results. The results consisted of the overall module marks, which included CA and examination. The ANOVA analysis was initially completed on the first three student cohorts (from 2012 – 2014). The AVOVA analysis investigated if the introductory programming module, without the additional Scratch module, produced statistically similar results each year (2012 – 2014). If it did not, measurement of the impact or value of the additional Scratch module may prove difficult to justify, due to the variance between the initial three student cohorts. One could argue that if the fourth year's results increased or decreased, this may not be due to the inclusion of Scratch but corresponded to the variance found in the previous years. The results of the ANOVA analysis for the 2012 – 2014 years is presented in Table 3.

ANOVA Analysis: 2012-2013, 2013-2014, 2014-2015	
Total Students	83
Total Mean	70.8223
Total Std. Dev.	17.3318
f-ratio	0.1711
p-value	0.8426

Table 3: ANOVA analysis of student performance from the first three year groups

Table 3 reports no statistical difference was found between each of the first three year groups, with a *p value* > 0.05. Table 4 presents the results of the ANOVA analysis on all four year groups.

ANOVA Analysis - All four year groups	
Total Students	113
Total Mean	69.8606
Total Std. Dev.	16.3456
f-ratio	0.4851
p-value	0.6932

Table 4: ANOVA analysis of student results from all four year groups

Table 4 reports no significant difference between all of the four year groups with a *p value* > 0.05.

5.2 Qualitative Analysis

This section reports on a survey that was administered at the end of academic year. The survey was conducted about one month before the end of the academic year. The survey was completed using the institutions standard end of year feed-back survey (66% of students took part). The Figures presented below, are questions relative to this study, taken from the institutions standard end of year feedback survey. Scratch was not specifically mentioned in this survey, thus any mention of it was unprompted. If Scratch was therefore mentioned it must have had some form of positive or negative effect on the students.

Initial Survey – SV1

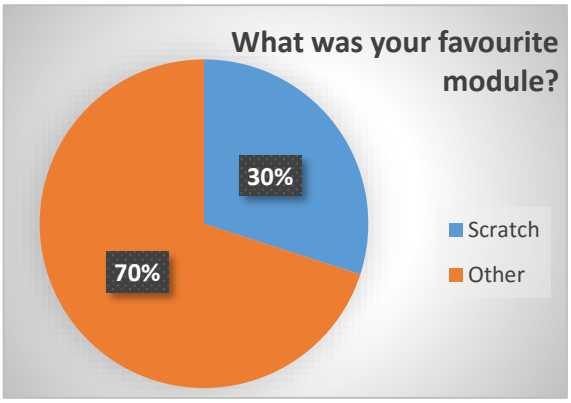

Figure 1: SV1

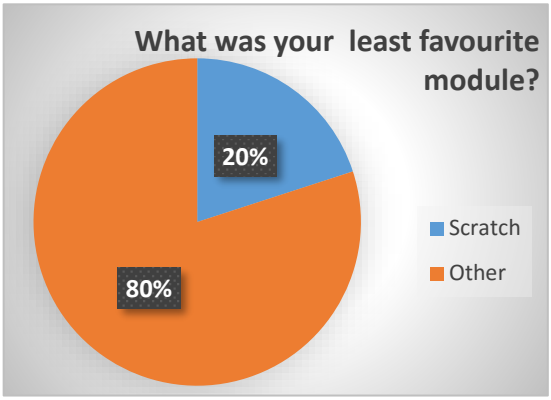

Figure 2: SV1

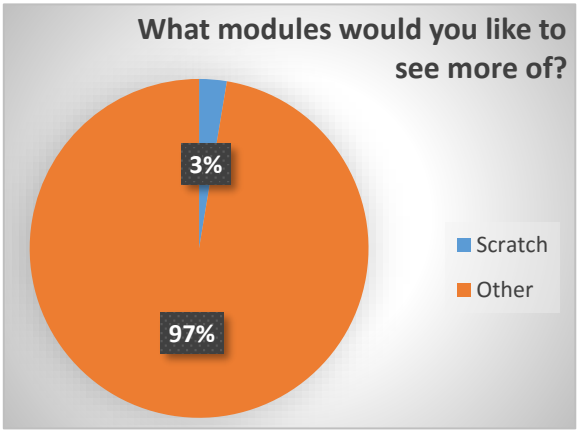

Figure 3: SV1

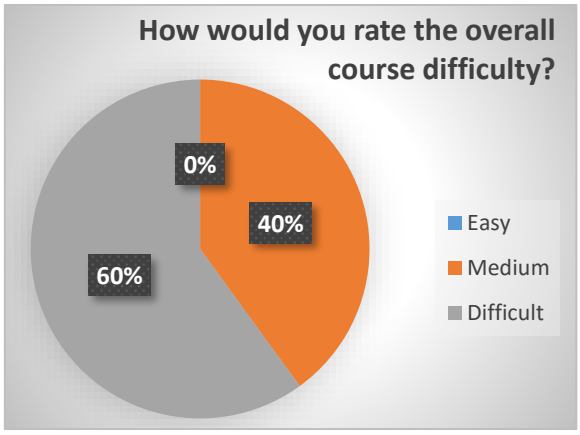

Figure 4: SV1

6. Conclusions and Future Work

6.1 Quantitative Analysis

Self-Efficacy

The results presented in Table 2, report no significant differences in programming self-efficacy. This finding is noteworthy as this may suggest that programming self-efficacy is not the cause of the higher motivation that was reported by Kereki (De Kereki, 2008). Anecdotally this may be due to the "fun" nature of Scratch or that the programs created in Scratch are more palpable or visual than a console application.

Programming Success

The results presented in Table 3, showed that over the first three years of the study, the variance of the student results were not significantly different with a p-value > 0.05. Thus the course produced similar results over the initial three years examined. This finding was significant as when the additional Scratch course was introduced, if a significant difference in results is then found, this would suggest that Scratch had an effect on the fourth year group's results, be it a positive or negative effect. When the fourth year group concluded after been exposed to both the staple introductory programming module and the additional Scratch based module, an additional analysis of variance was conducted on all of the four year groups. Table 4 presented a p-value > 0.05, concluding that no significant statistical difference in the variance of results was found between any of the four year groups. Section 6.3 discusses a noteworthy underlying phenomena that may have contributed to this finding, and may be worth further investigation.

6.2 Qualitative Analysis

Survey

The information presented in Figure 1 suggests that a large cohort of students really liked Scratch. This is a significant finding, as mentioned in Section 3, the overall course consisted of a multitude of diverse modules (ten in total). This was to give students a flavour of computer science. The fact that students were unprompted about Scratch but selected it as their favourite module, we believe to be very significant and positive regarding its use alongside CS1. Figure 2 suggests that some students did not like Scratch. This may require further investigation as the qualitative survey was administered at the end of the academic year. This may have biased their feedback, possibly due to "outgrowing" Scratch, after they have mastered subjects like OOP or EDP, thus a survey conducted at the end of the delivery of the CS1 and Scratch module may

prove beneficial in understanding this finding. A point that was noted when examining the survey, was that the overall favourite module of students was Event Driven Programming, using a graphical user interface. It may be argued that this module is too not dis-similar to Scratch, it could also be the case that a more palpable or visual output from a program is more desirable by students.

In Figure 4, students reported the overall course to be medium to difficult, with the majority of students reporting difficult, which is agreement with other research (De Kereki, 2008). Figure 3 suggests that the amount of Scratch delivered was sufficient, thus inferring that additional exposure to Scratch may not be productive, and possibly contributing to the negative feedback findings towards Scratch found in Figure 2.

Additional Qualitative Work

The authors contacted the student cohort after the course was complete to conduct additional qualitative work. This consisted of informal communication, asking more specific questions. Only a small number replied (4 students), but the feedback was noteworthy. All the students responded that Scratch helped with difficult concepts such as iteration and conditional statements which agrees with Kereki, in that students strongly believe that Scratch helps them with difficult concepts (threshold concepts) early on (De Kereki, 2008). All students reported that C# was more enjoyable, but again this feedback was after they had mastered other programming subjects. This may confirm that Scratch although useful in the early stages, students may outgrow it, and prefer languages like C# or Java. Student quotes further assert this: "*Scratch helped make the ideas around nested loops and if statements a lot clearer to me, in comparison to even diagrams.*" and "*I felt it definitely simplified the concepts. With scratch, it was a lot easier to see the loops and if statements in a way that was laid out simply and made sense.*"

6.3 Overall Conclusions and Future Work

This study concludes that Scratch does not increase student programming self-efficacy when delivered alongside CS1. The study also concludes that Scratch delivered in parallel to CS1, does not increase student performance. This may not bode well for the use of Scratch alongside CS1, but upon further investigation, a very significant finding, showed a substantial variance in the average module pass rates between the first three year groups and the final year group which was examined in this study. The average module pass rate is calculated across all modules delivered in the academic year (ten in 2015-2016 which included the additional games module

and nine in the three previous years 2013 – 2015 where Scratch was not included). The average module pass rates for each year group is presented in Table 5.

	2012-2013	**2013-2014**	**2014-2015**	**2015-2016**
Average pass rate over all modules X	81.51%	77.83%	70.45%	46.37%
Std. Dev. (Pass Rates over all Modules) σ	13.71%	19.65%	17.44%	18.74%

Table 5: Average overall module pass rates from each year group

Initially an ANOVA analysis was completed on the initial three years (2012-2014) concluding that no statistical difference was found between the average pass rates of each delivered module, with a with a *p value* of 0.3916. Next an ANOVA analysis was completed on all four year groups, resulting in a significant difference found in the average module pass rates with a *p value* of 0.0004. A Tukey HSD post-hoc analysis confirmed that it was the 2015-2016 results which were significantly statistically different to the other three year groups. This suggests that the 2015-2016 student cohort was overall, significantly weaker than the previous three cohorts. Given this, the CS1 performance results were statistically similar to that of the previous three years, thus it could be inferred that while the 2015-2016 group was weaker, the cohort performed as good as the previous three years in the CS1 module. This may be attributed to the additional Scratch gaming module. Future work in this space, based on this finding, strongly justifies another study to confirm this hypothesis. The authors also suggest that multiple qualitative surveys be administered at various times through-out the year, to further investigate feedback on the use of Scratch and students perceptions of it as the course progresses.

References

Armoni, M., Meerbaum-Salant, O., & Ben-Ari, M. (2015). From Scratch to “real” programming. *ACM Transactions on Computing Education*, *14*(4), 25:1–25:15. https://doi.org/10.1145/2677087

Bergin, S. (2006). A Computational Model to predict programming performance, (May).

Bergin, S., & Reilly, R. (2005). *Programming: factors that influence success*. *ACM SIGCSE Bulletin* (Vol. 37). https://doi.org/http://doi.acm.org/10.1145/1047124.1047480

Bergin, S., & Reilly, R. (2006). Predicting introductory programming performance: A multi-institutional multivariate study. *Computer Science Education*, *16*(February 2015), 303–323. https://doi.org/10.1080/08993400600997096

Campbell, J., Horton, D., & Craig, M. (2016). Factors_for_success_in_online_.PDF. *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, 320–325.

Cherenkova, Y., Zingaro, D., & Petersen, A. (2014). Identifying Challenging CS1 Concepts in a Large Problem Dataset. *Sigcse ’14*, 695–700. https://doi.org/10.1145/2538862.2538966

De Kereki, I. F. (2008). Scratch: Applications in Computer Science 1. *Proceedings - Frontiers in Education Conference, FIE*, 7–11. https://doi.org/10.1109/FIE.2008.4720267

Hegarty-kelly, E., Lockwood, J., Bergin, S., & Mooney, A. (2015). A roadmap to create online videos to facilitate student learning of threshold concepts. In *International Conference on Enguaging Pedagogy (ICEP)*.

Horn, R. a. (2008). Understanding the One-Way Anova, 1–13. Retrieved from http://oak.ucc.nau.edu/rh232/courses/EPS525/Handouts/Understanding the One-way ANOVA.pdf

Kalelioğlu, F., & Gülbahar, Y. (2014). The Effects of Teaching Programming via Scratch on Problem Solving Skills, *13*(1), 33–50.

Keenan, M. (2013). Up to 4,500 jobs unfilled due to skills shortage in IT. Retrieved September 13, 2015, from http://9thlevel.ie/2013/05/23/up-to-4500-jobs-unfilled-due-to-skills-shortage-in-it/

Kelly, J. O., Mooney, a, Ghent, J., Gaughran, P., Dunne, S., & Bergin, S. (2004). An Overview of the Integration of Problem Based Learning into an existing Computer Science Programming Module 2 Overview of our PBL Implementation. *Problem-Based Learning International Conference 2004: Pleasure by Learning*.

Liston, M., Frawley, D., & Patterson, V. (2016). A Study of Progression in Irish Higher Education, 23–24.

Lockwood, J., Mooney, A., & Bergin, S. (2016). A neurofeedback system to promote learner engagement, (1), 1–12.

Malan, D. J., & Leitner, H. H. (2007). Scratch for Budding Computer Scientists. *Sigcse 2007: Proceedings of the Thirty-Eighth Sigcse Technical Symposium on Computer Science Education*, 223–227. https://doi.org/10.1145/1227504.1227388

McGuire, E. (2015). Solving gender gap in tech will help fix skill shortage. Retrieved from http://www.irishtimes.com/business/work/solving-gender-gap-in-tech-will-help-fix-skill-shortage-1.2363518

Meyer, J. H. F., & Land, R. (2003). Threshold concepts and troublesome knowledge: Linkages to ways of thinking and practising within the disciplines. *Improving Student Learning – Ten Years On.*, *4*(1), 1–16. https://doi.org/10.1007/978-3-8348-9837-1

MIT media lab. (2005). Scratch - Imagine, Program, Share. Retrieved from https://scratch.mit.edu/

Mooney, A., & Bergin, S. (2013). A study on alternative strategies for sharing lecture notes using a VLE to promote in-class participation, *2013*.

Mooney, O., Patterson, V., O'Connor, M., & Chantler, A. (2010). *A Study of Progression in Irish Higher Education*.

Nolan, K., Mooney, A., & Bergin, S. (2015). Facilitating student learning in Computer Science : large class sizes and interventions. *International Confernce on Engaging Pedagogy*.

O'Brien, C. (2016). CAO countdown: Tech graduates in demand amid skills shortage. Retrieved from http://www.irishtimes.com/news/education/cao-countdown-tech-graduates-in-demand-amid-skills-shortage-1.2701990

Quille, K., & Bergin, S. (2015). Programming: Factors that Influence Success Revisited and Expanded. In *Programming: Factors that Influence Success Revisited and Expanded*. ICEP.ie. Retrieved from http://icep.ie/wp-content/uploads/2016/03/ICEP_2015_paper_16.pdf

Quille, K., & Bergin, S. (2016). Programming: Further Factors that Influence Success. In *Psychology of Programming Interest Group (PPIG), 7th to 10th Spetember, University of Cambridge.*

Quille, K., Bergin, S., & Mooney, A. (2015). PreSS #, A Web-Based Educational System to Predict Programming Performance, *4*(7), 178–189.

Stajkovic, A. (2016). Social cognitive theory and self- efficacy : Implications for motivation theory and practice . Social Cognitive Theory and Self . efficacy : Implications for Motivation Theory and Practice are, (March).

Vivian, R., Falkner, K., & Falkner, N. (2013). Computer Science students' causal attributions for

successful and unsuccessful outcomes in programming assignments. *13th Koli Calling International Conference on Computing Education Research (Koli Calling)*, 125–134. https://doi.org/10.1145/2526968.2526982

Watson, C., Li, F. W. B., & Godwin, J. L. (2014). No tests required: comparing traditional and dynamic predictors of programming success. *Proceedings of the 45th ACM Technical Symposium on Computer Science Education - SIGCSE '14*, 469–474. https://doi.org/10.1145/2538862.2538930

Wiedenbeck, S. (2005). Factors affecting the success of non-majors in learning to program. *Proceedings of the 2005 International Workshop on Computing Education Research - ICER '05*, 13–24. https://doi.org/10.1145/1089786.1089788

www.code.org. (n.d.). Promote Computer Science. Retrieved from https://code.org/promote