STEAM WORKS: Student coders experiment more and experimenters gain higher grades

Matthew Yee-King, Mick Grierson and Mark d'Inverno
Department of Computing
Goldsmiths, University of London, UK

***Abstract*—For the last decade, there has been growing interest in the STEAM approach (essentially combining methods and practices in arts, humanities and social sciences into STEM teaching and research) to develop better research and education, and enable us to produce students who can work most effectively in the current and developing market-place. However, despite this interest, there seems to be little quantitative evidence of the true power of STEAM learning, especially describing how it compares and performs with respect to more established approaches. To address this, we present a comparative, quantitative study of two distinct approaches to teaching programming, one based on STEAM (with an open-ended inquiry-based approach), the other based on a more traditional, non-STEAM approach (where constrained problems are set and solved). Our key results evidence how students exhibit different styles of programming in different types of lessons and, crucially, that students who tend to exhibit more of the style of programming observed in our STEAM lessons also tend to achieve higher grades. We present our claims through a range of visualisations and statistical validations which clearly show the significance of the results, despite the small scale of the study. We believe that this work provides clear evidence for the advantages of STEAM over non-STEAM, and provides a strong theoretical and technological framework for future, larger studies.**

***Keywords*—STEAM; xAPI; coding; education; pedagogy**

I. Introduction

Over the last 10 years, we have been central to the development and delivery of a range of degree programmes at Goldsmiths that aim to bring an arts inflected pedagogy into the teaching of computer science. These include music computing, digital arts computing and creative computing at undergraduate level then computational studio arts at postgraduate level, and we make extensive use of what is considered a STEAM approach to engineering education[1]. Over the last decade, we have observed how a range of employers, especially from the creative industries, are keen to employ graduates from our courses as they demonstrate a creatively driven approach to engineering and are able to adapt quickly to using new technologies in new settings.

However, despite the apparent success of these courses and the pedagogical approach taken, we have not as yet been able to empirically evidence any specific impact upon the way students learn. We have set out to address this large gap in our knowledge by developing and trialling a suite of educationally focused, social media type tools with learning analytics baked in. ([1], [2], [3]). Codecircle[2] is the most recent of these tools and is the system used for the research described in this paper. It is a browser based, integrated development environment, a full technical description of which can be found in [4]. The system is being used by hundreds of students in our department and its built in data gathering functionality makes it possible to conduct quantitative analyses of usage patterns. In this paper, we use codecircle to examine and compare the activities of students when they are exposed to STEAM and non-STEAM style lessons.

We believe it is important to produce clear evidence on how people learn programming, and how different pedagogical approaches are most effective and have developed tools that enable us to do so. We are especially interested in making the learning of programming available to everyone because coding has been highlighted by industry leaders as 'the red thread that runs through Europe's future professions' [5]. To maximise the number of 'computational thinkers' [6] entering the job market, more people, from a much greater range of backgrounds and disciplines, will need to learn how to code. We need to ensure that the methods we use to teach coding are both inclusive and effective. This is not an easy task - Margolis and Goode summarise the challenge of developing inclusive CS education as follows: "The goal is to bring the students to the subject in a way that allows them to understand it deeply and make it part of their own experience without watering down the content or neglecting the fundamental concepts and modes of inquiry that characterize the discipline" [7].

We think that the prevailing approach to teaching programming, which seems to be largely derived from classical engineering education, is not the most effective way to attract and educate a diverse group of new coders who can function effectively in the modern workplace. This 'non-STEAM' approach typically involves students studying a large body of pre-existing technical knowledge and learning how to apply it deductively to constrained problems that are designed to test this knowledge. STEAM offers an alternative approach involving an inductive, exploratory process driven by self-defined goals, more akin to that seen in creative arts education.

The question we wish to consider is: do these distinct

[1]strictly STEAM = STEM + Arts, but we do not exclude input from other non-STEM disciplines.

[2]https://live.codecircle.com

approaches actually impact on the way a student goes about programming and, if so, is one approach better than the other? In this paper, we shed some light on these questions by describing the results of a study wherein students worked on STEAM and non-STEAM style programming activities. In particular, we address the following research questions: Do students code differently when undertaking STEAM and non-STEAM exercises? Do students report qualitatively different experiences when working on different types of exercises? Is there a relationship between coding behaviour and final grades? By exploring these questions, this paper makes three main contributions:

1) A comparative, quantitative analysis of student programming behaviour in STEAM and non-STEAM lessons.
2) A clear definition of STEAM and non-STEAM pedagogy with specific examples of lessons using both.
3) A reusable experimental and technological framework within which it is possible for researchers to conduct a range of computer science education studies.

Our results support anecdotal evidence from teachers and lecturers that experiential learning, such as that found in arts education and STEAM, is a critical component of successful, deeper STEM learning.

The paper is organised as follows: in the following section, we will introduce some previous work that describes STEAM and non-STEAM approaches to teaching coding and which analyses student coding behaviour. In section III we describe the experimental and technological framework we have developed to enable studies into computer science education. In section IV we will describe the method used for this particular study. In section V we will describe and analyse the data that resulted from the study. In section VI we will discuss and evaluate the results, concluding in section VII.

II. Background

In this section we will explain what we mean by STEAM and non-STEAM pedagogy, based on references to the literature, then we will discuss some previous studies which analysed student coding behaviour.

A. STEAM and non-STEAM

Classical STEM teaching, which is sometimes characterised as 'chalk and talk', lecture based learning, is still prevalent in undergraduate engineering education. In 2015, Connor et al stated that 'Developments in student-centric learning such as problem-based and project-based learning have so far had relatively little impact on mainstream engineering education' [8]. This non-STEAM approach involves a well established and substantive body of knowledge about an engineering subject being explained to students, who are then required to deductively apply it to carefully designed and constrained problems. This approach is perhaps most clearly seen in assessments, which 'tend to focus on whether knowledge or skills have been obtained', in a Thorndikian manner [9]. Non-STEAM project work can be characterised as tutors setting carefully constrained projects with goals aligned with the pre-existing ontology of knowledge about the subject. The learner is given limited autonomy about how they approach a project and what its goals are [10].

STEAM, on the other hand, offers an arts inspired approach to engineering education where students are encouraged to construct their own ontologies of understanding through an active process of creation [11]. It is clearly based on a constructivist theory of learning, after Piaget, but another influence is Dewey. Assessment is a qualitative, formative process, based around Deweyian 'creative feedback' and an appreciation of the self organisational characteristics of a deeper learning process [12]. Further insight into STEAM can be gained from Rose and Smith who state: 'the STEAM agenda should be about deep, sustained, powerful engagement as a way of learning' [13]. This leads to a quite open ended, project based approach to learning which certainly pre-dates the acronym STEAM, and which has been previously discussed in a science education context by researchers such as Gallagher et al., and Papert [14], [15]. Perhaps STEAM has arisen as a result of the growing recognition of the lack of constructivist learning in computer science education, as noted by Ben-Ari, who contrasts this with its prevalence in Mathematics and Science [16].

Thus we find that the character of STEAM education appears well understood, but crucially there is a lack of studies which attempt to quantify its nature or to compare it directly with the traditional STEM approach. This is not surprising - in a recent review of computing education research (CER) literature, Malmi et al. noted a general lack of theory driven research in CER, and indeed in engineering education research in general [17].

In response to calls seen in the literature, we have designed the study reported here based on a theoretical perspective upon STEAM learning. We also describe a re-useable, experimental framework within which the theory can be practically investigated. Finally, our data driven approach makes use of standards and approaches being developed in the fields of learning analytics and educational data mining [18].

B. Analysing coding behaviour

We shall now consider some examples of work that describes and analyses student programming behaviour. Rodrigo and Baker looked at programming behaviour, such as repeated attempts to compile the same code [19]. They developed a model that was able to detect affects in programming labs, such as frustration. The data we use in our study is higher resolution, allowing us to examine coding behaviour at the keystroke, rather than the compile event level. Blikstein et al. describe a range of metrics that can be automatically extracted from IDE code snapshots, and use them to characterise student coding behaviour [20]. They linked their classifications to Papert's tinkerer and planner categories [15].

Mahadevan et al. describe a STEAM oriented coding environment wherein students learn Javascript and Python by computationally re-arranging chunks of audio (remixing) in

a web based system called EarSketch[21]. EarSketch is now being trialled at scale with thousands of high school age children based around a carefully designed STEAM curriculum which aims to match the learning goals of the equivalent non-STEAM curriculum. Our work differs in this phase in that we are focused on fine grained details of student coding behaviour and how this changes with changing pedagogy.

III. Experimental and technological framework

In this section, we will describe the experimental and technological framework we have developed in order to carry out comparative investigations of different pedagogical styles in computer science education. The framework consists of 1) an education focused integrated development environment (IDE) 2) learning analytics and 3) a reusable experimental method.

A. The educational IDE

We have developed an educational, browser based IDE which allows students to write programs in their web browser [4]. The system is currently being used in a range of our teaching, including MOOCs and various on campus courses. A screen shot can be seen in figure 1. The key features of the system are as follows:

1) Web browser based.
2) Live coding where code is re-interpreted as you type.
3) High resolution, timestamped code editing logs.
4) Programs are written in Javascript.
5) Jshint highlights basic coding errors.
6) integrated audiovisual libraries for scaffolded use of real-time graphics and sound.
7) Real time, collaborative coding via an operational transformation engine [22].

The key feature from the above list for the experimental framework is the high resolution code editing log data. It describes all edits made to the code at the keystroke level, with timestamps, and the content of the edit. Interestingly, this data comes as a side effect of the operational transformation (OT) engine which underlies the code editor. This engine is there to enable real time collaborative coding, similar to Google Documents. It is implemented using Gentle's sharejs library which grew out of the Google Wave project [23]. OT provides a set of algorithms which work together to provide the best possible version of a document that has been simultaneously edited by multiple editors. The operations can also be re-run such that the process of creating the document can be observed step by step. We have not fully explored the educational experimental possibilities of this logging system yet, and in the work presented here, we use it as a means of gathering high level statistics about the types of code edits students were making during the lessons.

B. Learning analytics

The second part of our experimental framework is a learning analytics system. The code editing logs are converted from their raw form, generated by the OT engine, into a semantic form according to the xAPI specification [24]. Briefly, xAPI provides a formal way of describing interactions between learners and learning technology as a set of xAPI statements in JSON format. We have defined a set of xAPI statement types which are suitable for describing the activities of programmers, and published it as a github repository[3] as we found that there were no pre-existing 'recipes' for such. The main elements of a statement are shown below:

```
{
"actor" : who acted?
"verb" : what did they do?
"object" : what were they interacting with
"timestamp" : when?
"result" : what was the outcome?
"context" : what was the lesson/ activity?
}
```

The verbs we selected to describe coding actions in this study were 'create' (new document), 'insert' (code to document), 'delete' (code from document) and 'terminate' (a code statement). Terminating a code statement means inserting code that ends with a semicolon. As our work develops in the future, we can define further statement types, for example, we might want to log when coders use control flow constructs like loops, when they use library functions or when they access online documentation.

The xAPI statements describing all the logged actions are stored in a Learning Record Store (LRS), which is a web application specifically designed to store xAPI statements for the purposes of applying learning analytics to them. We developed our learning analytics to talk to the LRS and to process xAPI statements.

To make accessing and browsing our xAPI statements easier, we developed a dashboard that connects to the LRS to retrieve and visualise xAPI statements, as shown in figure 2. With this system, it is possible to rapidly explore live data on the system, for example viewing all actions by a certain user, in a single document or within a particular lesson.

C. Experimental design

Having introduced our education and analytics technology, we will now describe our general approach to experimental design. In essence, a set of students participate in a set of lessons which include programming activities with different characteristics (described in more detail below). The characteristics that can vary might be the structure of the learning activity, or the configuration of the IDE. The students are asked to complete short surveys after every lesson wherein they self-report their experiences in the lessons. The analytics system gathers data which is connected to lessons via tagging and time stamping. The experimental design aims for high ecological validity [25] - these are real lessons and they are designed to be effective and enjoyable. We rely on the high resolution data gathering to provide us with robust quantitative results.

[3]https://github.com/yeeking/xapi-coding-recipes

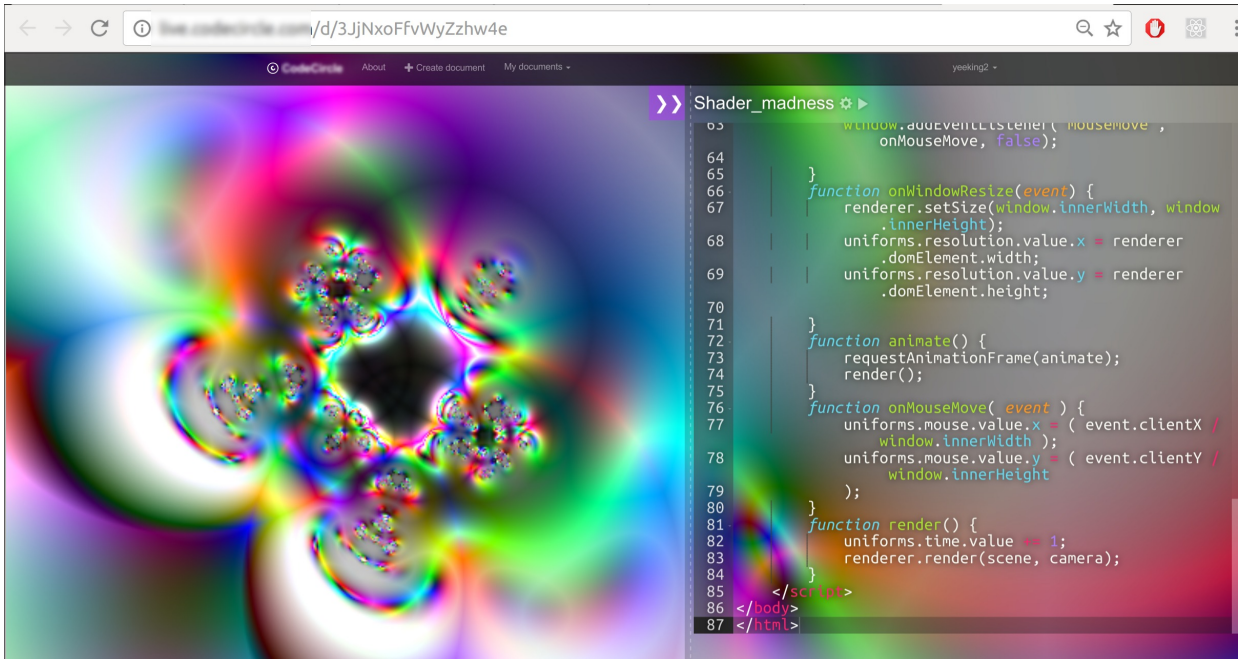

Fig. 1: Screen shot of the IDE. The code for the program can be edited on the right. The output of the program can be seen on the left. The code editing panel is made opaque when editing so the code can be clearly seen.

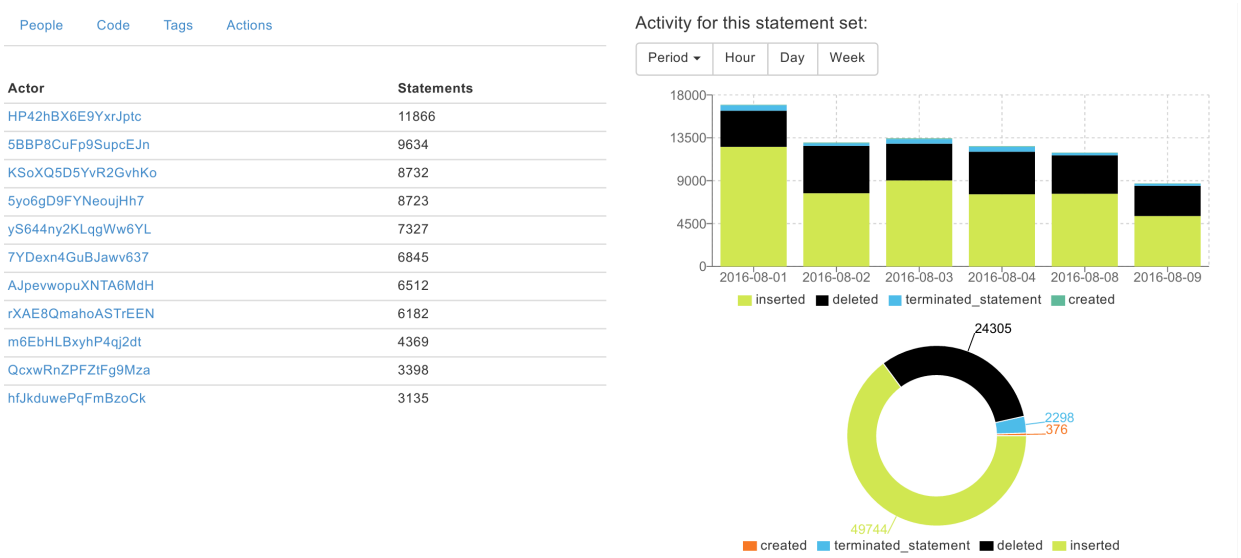

Fig. 2: Screen shot of the learning analytics dashboard. On the left is the data browser, currently showing an anonymised list of users, on the right are two visualisations of coding behaviour over the 6 days of the study.

IV. The Study

Having described our general technological, experimental framework, we will now describe a specific instance of an experiment that we have carried out within the framework which aims to compare programming behaviour and student experience in STEAM and non-STEAM style lessons.

The study involved 11 undergraduate, arts computing students at a summer school. They participated in 12, 2 hour, group lessons over two weeks, wherein 6 activities were STEAM and 6 were non-STEAM. There were two lessons in a day, one in the morning and one in the afternoon, one STEAM and one non-STEAM. Sometimes STEAM was in the morning, sometimes non-STEAM. The activities are described in more detail below. All lessons were taught by the same tutor and involved a short presentation by the tutor followed by students working on a programming activity. Finally, all of the students had beginner to intermediate level programming skills - we knew this because they had been using the programming environment (described above) for 2 weeks prior to the study

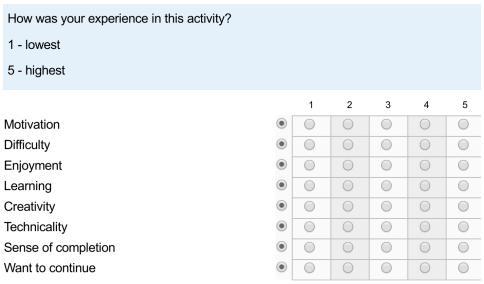

Fig. 3: Screenshot of how the survey looked in our VLE.

and this allowed us to estimate their programming experience level.

All 12 lessons were based around learning to manipulate audio and graphics using Javascript, so both STEAM and non-STEAM lessons involved a multimedia component. We are taking a philosophical position here - just because a lesson involves graphics or sound, it does not make it STEAM. STEAM is a specific approach to education which is visible in the nature of the learning activities. The programming activities in the lessons fell into 4 categories: Fill in the gaps (FITG/ non-STEAM), Implement a specification (SPEC/ non-STEAM), Work to aesthetic goals (WTAG/ STEAM) and Fork and customise (FAC/ STEAM). In FITG, students were provided with an incomplete program and they had to fill in the gaps. This is a typical non-STEAM style teaching method as they were provided with a contrained problem that required that they engaged with very specific engineering techniques. In SPEC, students were given a simple list of requirements that they had to implement in a program; it was slightly less constrained than FITG but was still designed to engage them with specific techniques. In WTAG, students were encouraged to engage with an aesthetic concept such as timbre or motion and to develop an idea for a simple program that explored that concept. This was STEAM as their exploration was open ended and driven by their own interests. The FAC lessons involved students browsing through eachothers' work and selecting something they would like to customise. This was STEAM as they were not forced to customise in any particular way, but to be driven by their own ideas about what they should do.

After each activity the students completed a short survey, where they rated their experience from 1 to 5 for: 'motivation', 'difficulty', 'enjoyment', 'learning', 'creativity', 'technicality', 'sense of completion' and 'want to continue'. We selected these measures based on consideration of what might allow differentiation between STEAM and non-STEAM. For example, one might imagine that students who had to fill in some gaps in a program (non-STEAM) would feel a stronger sense of completion than students who were asked to work to aesthetic goals (STEAM), or that students who had set their own project goals (STEAM) might be more motivated to continue.

TABLE I: P-values for variation between STEAM and non-STEAM lessons per self reported experience metric.

Metric	p-value
Motivation	0.533
Difficulty	0.7
Enjoyment	0.862
Learning	0.674
Creativity	**0.037***
Technicality	0.707
Sense of completion	0.407
Want to continue	0.927

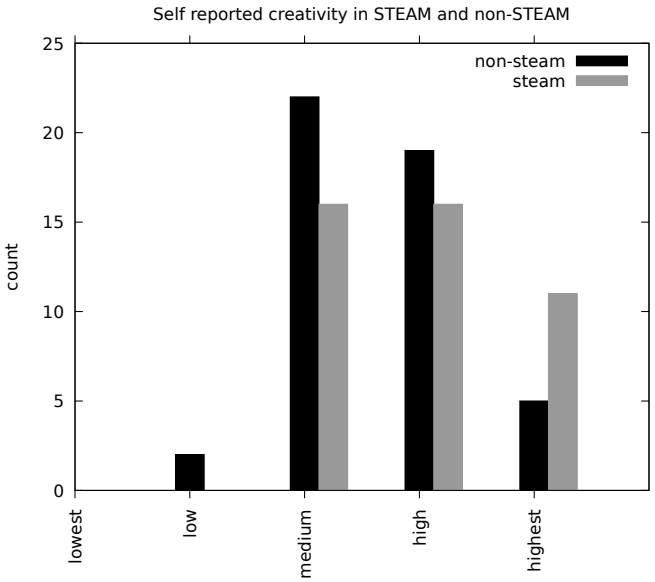

Fig. 4: Histogram of the two distributions of self reported creativity, showing the tendency for students to report higher levels of creativity in STEAM.

V. Results and analysis

The code logs and survey results form the raw data for the analysis below. The total number of code editing operations logged by the system for all students in the 12 activities was 77,923 and 91 surveys were completed (out of a possible 132). An example of how the survey looked in our VLE is shown in figure 3.

The main thrust of our analysis is to separate the data into two sets: those data resulting from STEAM lessons and those resulting from non-STEAM lessons. This provides us with two experimental conditions between which we can compare student experience and coding behaviour. We will describe three analyses in the following sub sections: self reported experience metrics, coding activity ratios and final grade correlation with activity ratios.

A. Self reported experience

In our first analysis, we measured the significances of the variation between the answers to the post STEAM and non-

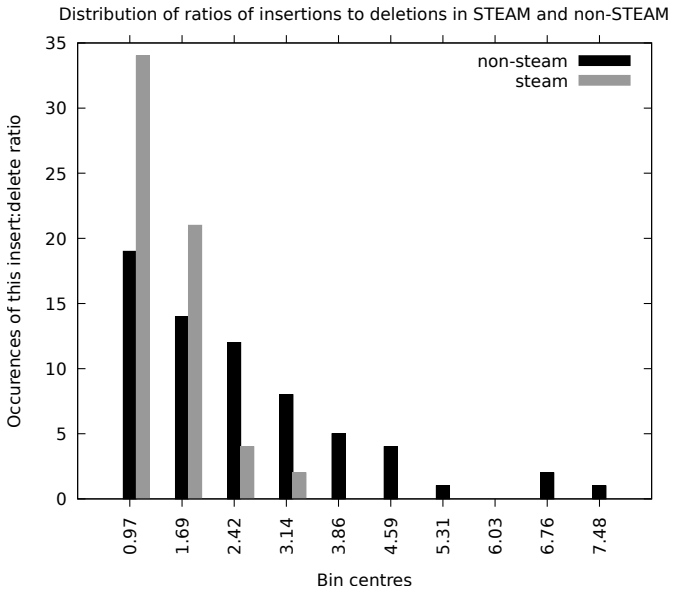

Fig. 5: Distribution of insert:delete ratios in STEAM and non-STEAM. A value of 1 means the same number of deletions and insertions happened in a lesson. A value > 1 means there were more insertions than deletions.

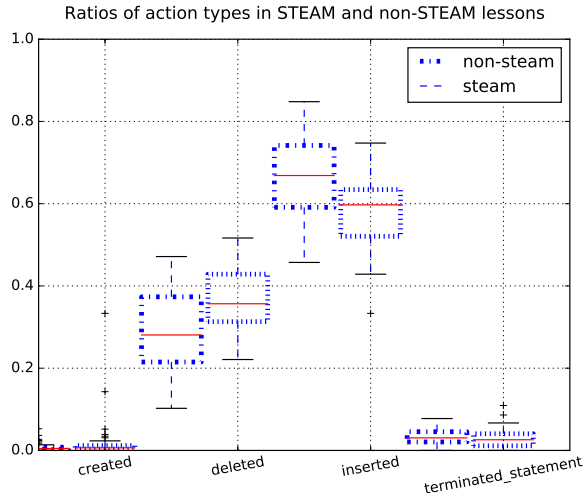

Fig. 6: Ratios of code operation types in STEAM and non-STEAM lessons. Each box shows the range of values for a verb in a lesson type. High up boxes indicate more common actions, box size indicates the range of values observed.

STEAM activity questions using a Wilcoxon signed-rank test which is appropriate for estimating non-parametric effect size. We used non-parametric statistics since we could not assume that the answers followed any particular distribution. As mentioned above, we selected experience questions which we thought might differentiate between the two lesson styles, or more formally, our aim was to attempt to nullify the hypothesis that students would report having the same experience in both lesson styles.

The p-values obtained from the test are shown in Table I. The only metric that was significantly different was the students' reported experience of creativity - participants reported higher levels of creativity for the STEAM lessons. The distributions of self reported creativity levels the lessons are shown in figure 4 and it can be seen that the ratings tended to be higher in the STEAM lessons. We will discuss this result, and the lack of significant variation in the other metrics in the discussion section.

B. Coding activity ratios

In our second analysis, we looked at the codecircle logs which had been converted into xAPI statements. We computed ratios between the types of actions (verbs) observed in the STEAM and non-STEAM datasets, one ratio for each student in each lesson. For example, given our verb set of create-insert-delete-terminate, a ratio of $0.1 : 0.3 : 0.5 : 0.1$ would indicate that in that lesson, the code logs consisted of 10% creations, 30% insertions, 50% deletions and 10% terminations.

The range of ratios observed across all the lessons is shown in a box plot in figure 6, where each box shows the spread of values for a verb's ratio across all lessons of a particular type. A visual inspection of this graph suggests that there were more insertions relative to deletions in all lessons, but this was less pronounced in STEAM, where there seemed to be relatively more deletions happening. To verify this, we calculated two sets of values: the insert:delete ratios for all students in all STEAM and the insert:delete ratios for all students in all non-STEAM lessons. This gave us two sets of values of length 66 (11 students, 6 lessons), where each value represented the ratio between insertions and deletions for one student in one lesson. E.g. a value of 7 means there are 7 inserts for every delete. A value of 0.5 means there are 2 deletes for every insert. We used a two sided Kolmogorov-Smirnov test which is a non-parametric test suitable for testing the null hypothesis that two continuous valued samples are drawn from the same distribution. This would allow us to decide if the visually apparent difference between the two sets of ratios was significant. It yielded a p-value of 7.747e-6, which is very significant. Therefore, the null hypothesis can be rejected - insert:delete ratios *are* different in STEAM and non-STEAM lessons. A histogram showing the distribution of insert:delete ratios is shown in figure 5. It it clear that the STEAM lessons tend towards parity between insert and delete operations (values close to 1) whereas non-STEAM lessons tend to have more inserts than deletes (values higher than 1).

C. Correlating final grade with activity ratios

In the next analysis, we follow up on the observation of greater deletion behaviour in STEAM lessons and ask the question: is deleting code something that successful students tend to do more? We do this by examining the final grades achieved by students and their relationship with the students' individual delete:insert ratios. After the 12 lessons, the stu-

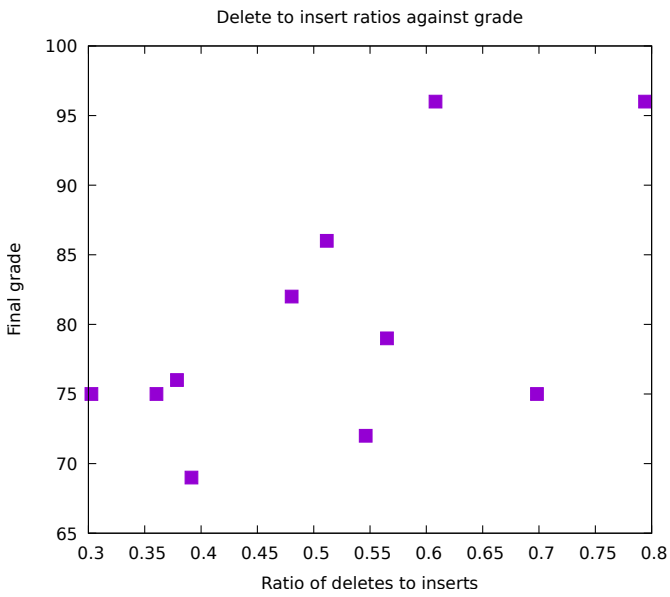

Fig. 7: Final grades achieved by students plotted against their ratio of deletion to insertion edits across all lessons. A value of 0.5 on the axis means there were twice as many insertions as deletions, a value of 1 on the x axis means there were the same number of insertions and deletions.

dents carried out a project of their own devising, which was graded by two tutors based on a presentation and on the technical and aesthetic quality of the work. A scatter plot showing grades plotted against the ratio of deletions to insertions (for coding done during the lessons, not the project) is shown in figure 7. Visual inspection suggests a positive correlation between the grade and the delete:insert ratio - students with high grades seem to do relatively more deletions. The Pearson correlation coefficient was 0.612 with a p-value of 0.046 - a significant, positive correlation. We also measured correlations of other metrics with the final grade: total number of edits, number of inserts and deletes. This yielded values of -0.179, -0.304 and 0.096 respectively, all much weaker correlations, some negative. This suggests that sheer number of edit operations does not predict the final grade. Therefore we can conclude that in this somewhat small data set, more successful students tended to do relatively more deletions.

VI. Discussion

We shall now return to the research questions posed at the beginning of the paper and view them through the lens of the evidence we have presented in the previous section.

a) Do students code differently when undertaking STEAM and non-STEAM exercises?: We have defined 4 lesson types, 2 STEAM and 2 non-STEAM lessons. We gathered detailed logs of student coding activity in multiple examples of these lessons. Our data suggests that the coding behaviour patterns of the same set of students varied significantly between STEAM and non-STEAM activities. Specifically, the ratio of insert to delete operations logged in the code editor was closer to 1 in STEAM lessons and higher in non-STEAM lessons, so students were doing relatively more deletes in STEAM lessons.

We interpret this result as a possible sign that students were experimenting more in the STEAM lessons - they were trying things out then deleting them, they were exploring the problem space through trial and error. This is exactly the kind of activity we would want to encourage.

b) Do students report qualitatively different experiences when working on different types of exercises?: We asked students to complete experience surveys after taking part in a series of 12 lessons wherein they rated their experience on 5 point scales for motivation, difficulty, enjoyment, learning, creativity, technicality, 'sense of completion' and 'want to continue'. We designed these scales based on our own intuition about which metrics might allow us to differentiate between the lesson styles. We were somewhat surprised to find that the lessons were not rated significantly differently on any of the metrics aside from creativity. Students rated their experience of creativity higher in STEAM lessons so students felt more creative in STEAM lessons. Since the other metrics did not vary between lessons, it seems possible to infer that this gain was obtained without a detrimental effect on those metrics.

c) Is there a relationship between coding behaviour and final grades?: Our final analysis was to look for relationships between the proposed "trial and error" behaviour which manifested in our data as higher delete to insert ratios and some sort of ground truth about student ability. In other words, did successful students carry out more trial and error? We were able to find a reasonably significant correlation between our trial and error metric and final grades achieved by students. People with higher final grades tended to have done more trial and error coding. The sheer number of edits made did not correlate at all with the grades achieved - the important feature was the nature of the editing behaviour. This is a really interesting result especially combined with the observation that STEAM lessons encouraged more trial and error behaviour in general across the cohort. Perhaps STEAM teaching can encourage students to develop better learning strategies involving greater experimentation. We would need to carry out a longer term study to gain a greater understanding of this suggestion.

VII. Conclusion

In this paper, we have described a study comparing classical, and STEAM approaches to computer science education. The study was motivated by our need to better evidence and describe the advantages and disadvantages of the arts inflected pedagogy that we have developed over the last 10 years or so at Goldsmiths. In order to carry out the study, we have developed an experimental and technological framework which makes it possible to carry out a range of studies into computer science education and specifically to look at how students approach programming in different learning contexts. It involves the careful design of lessons based on clear pedagogical theory,

intensive data gathering using a novel browser based programming environment, and the use of learning analytics and statistical methods to analyse and interpret the data, with an emphasis on a comparative approach.

The study reported in this paper provided some key results. First, students reported higher levels of creative experience in STEAM lessons but they did not report any undesirable reductions in the other areas of the experience. Second, their coding patterns were different, with relatively more delete operations in STEAM lessons. We interpret this as evidence of a more exploratory approach to programming with students much more open and confident about exploring the landscape through trial and error. Third, we observed that successful students tended to do relatively more deleting than unsuccessful students, suggesting that this trial and error approach is a successful strategy in learning to program.

In future work, we plan to improve and repeat the study with larger cohorts. Currently, we have several MOOCs with many thousands of users and run a range of on-campus courses at UG and PG level with many hundreds of students so have the perfect opportunity to do so. We also plan to develop new studies within our framework which will allow us to examine the effect of the other features of our IDE such as collaborative coding, live coding and audiovisual coding. We are also planning to expose the analytics to students, which would allow us to investigate student meta cognition, where students gain an understanding of their own learning process [26].

We believe that we have set the theoretical and experimental foundations for providing strong empirical evidence for the benefits of STEAM learning.

ACKNOWLEDGMENT

We would like to thank the students for taking part in the lessons and for agreeing to the data capture. Current development of the codecircle platform is funded through the Higher Education Funding Council for England's Catalyst scheme, under project code K31.

REFERENCES

[1] H. Brenton, M. Yee-King, A. Grimalt-Reynes, M. Gillies, M. Krivenski, and M. d'Inverno, "A Social Timeline for Exchanging Feedback about Musical Performances," in *British HCI Conference*, 2014, pp. 1–6.

[2] M. Yee-King, M. Krivenski, H. Brenton, and M. d'Inverno, "Designing educational social machines for effective feedback," in *8th International Conference on e-learning*. Lisbon: IADIS, 2014. [Online]. Available: https://eric.ed.gov/?id=ED557308

[3] M. Yee-King and M. d'Inverno, "Stimulating collaborative activity in online social learning environments with Markov decision processes Categories and Subject Descriptors," *Proceedings of the 9th International Conference on Educational Data Mining*, pp. 652–653, 2016.

[4] J. Fiala, M. Yee-king, and M. Grierson, "Collaborative coding interfaces on the Web," in *Proceedings of the 2016 International Conference on Live Interfaces*, 2016, pp. 49–58. [Online]. Available: http://www.liveinterfaces.org/proceedings2016.html

[5] Microsoft, Facebook, Liberty-Global, and Rovio, "Open letter to EU Ministers for Education Brussels,," 2014. [Online]. Available: https://www.microsoft.com/global/eu/RenderingAssets/pdf/2014 Oct 14 EU Code Week - European Coding Initiative Open Letter.pdf

[6] J. M. Wing, "Computational Thinking," *Communications of the ACM*, vol. 49, no. 3, pp. 33–35, 2006.

[7] J. Margolis and J. Goode, "Ten Lessons for Computer Science for All," *ACM Inroads*, no. 4, pp. 52–56, nov.

[8] M. Connor, S. Karmokar, and C. Whittington, "From STEM to STEAM : Strategies for Enhancing Engineering & Technology Education," *International Journal of Engineering Pedagogies.*, vol. 5, no. 2, pp. 37–47, 2015.

[9] N. Radziwill, M. Benton, and C. Moellers, "From stem to steaml reframing what is means to learn," *The STEAM Journal*, no. 1, sep.

[10] A. M. Connor, S. Karmokar, C. Whittington, and C. Walker, "Full STEAM ahead a manifesto for integrating arts pedagogics into STEM education," *Proceedings of IEEE International Conference on Teaching, Assessment and Learning for Engineering: Learning for the Future Now, TALE 2014*, no. December, pp. 319–326, 2015.

[11] D. Henriksen, "Full STEAM Ahead: Creativity in Excellent STEM Teaching Practices," *The STEAM Journal*, vol. 1, no. 2, feb 2014. [Online]. Available: http://scholarship.claremont.edu/steam/vol1/iss2/15

[12] M. d'Inverno and A. Still, "Creative Feedback: a manifesto for social learning," *In Proceedings of the Workshops held at Educational Data Mining 2014 conference, EDM 2014. London, UK*, 2014.

[13] C. Rose and B. K. Smith, "Bridging STEM to STEAM: Developing new frameworks for Art-Science-Design Pedagogy," *Rhode Island School District Press Release*.

[14] Gallagher, Shelagh A., William J. Stepien, B. T. Sher, and D. Workman, "Implementing Problem Based Learning in Science Classrooms," *School Science and Mathematics*, p. 136, 1995.

[15] S. Papert, *Mindstorms: children, computers, and powerful ideas*. Basic Books, Inc. New York, NY, USA, 1980. [Online]. Available: http://dl.acm.org/citation.cfm?id=1095592

[16] M. Ben-ari, "Constructivism in computer science education," *Journal of Computers in Mathematics and Science Teaching*, vol. 20, pp. 45–73, 2001.

[17] L. Malmi, J. Sheard, L. Malmi, J. Sheard, R. Bednarik, A. Korhonen, N. Myller, and A. Taherkhani, "Theoretical underpinnings of computing education research - What is the evidence ?" in *Proceedings of the tenth annual conference on International computing education research*, no. July, 2014, pp. 27–34.

[18] Z. Papamitsiou and A. A. Economides, "Learning Analytics and Educational Data Mining in Practice : A Systematic Literature Review of Empirical Evidence The research questions," *Educational Technology & Society*, vol. 17, no. 4, pp. 49–64, 2014.

[19] M. M. T. Rodrigo and R. S. Baker, "Coarse-grained detection of student frustration in an introductory programming course," *Proceedings of the fifth international workshop on Computing education research workshop - ICER '09*, p. 75.

[20] P. Blikstein, M. Worsley, C. Piech, M. Sahami, S. Cooper, and D. Koller, "Programming Pluralism: Using Learning Analytics to Detect Patterns in the Learning of Computer Programming," *Journal of the Learning Sciences*, no. 4, pp. 561–599.

[21] A. Mahadevan, J. Freeman, B. Magerko, and J. C. Martinez, "EarSketch: Teaching Computational Music Remixing in an Online Web Audio Based Learning Environment," *Proceedings of the Web Audio Conference (WAC)*, pp. 0–5, 2015.

[22] C. S. Ellis, "Operational Transformation in Real-Time Group Editors: Issues , Algorithms , and Achievements," in *Proceedings of the 1998 ACM conference on Computer supported cooperative work*, 1998, pp. 59–68.

[23] J. Gentle, "ShareJS - Live Concurrent Editing in your App," 2012. [Online]. Available: https://github.com/josephg/ShareJS

[24] ADLNET, "xAPI-Spec," 2016. [Online]. Available: https://github.com/adlnet/xAPI-Spec/blob/master/xAPI.md

[25] W. A. Sandoval and W. A. Sandoval, "Design-Based Research Methods for Studying Learning in Context : Introduction Design-Based Research Methods for Studying Learning in Context : Introduction," *EDUCATIONAL PSYCHOLOGIST*, vol. 39, no. 4, pp. 199–201, 2004.

[26] G. Biswas, J. S. Kinnebrew, and D. L. C. Mack, "How do students ' learning behaviors evolve in Scaffolded Open-Ended Learning Environments ?" in *Proceedings of the 21st International Conference on Computers in Education*, 2013.