Evidencing the Value of Inquiry Based, Constructionist Learning for Student Coders

https://doi.org/10.3991/ijep.v7i3.7385

Matthew John Yee-King(✉), Mick Grierson, Mark d'Inverno
Goldsmiths, University of London, UK
m.yee-king@gold.ac.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) with its potential to deliver better research and education, and to enable us to produce students who can work more 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-driven, inductive approach), the other based on a more traditional, non-STEAM approach (where constrained problems are set and solved deductively). 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

1 Introduction

Over the last 10 years, we have developed and delivered a range of degree programmes at our institution that aim to bring an arts inflected pedagogy into the teaching of computer science. We make extensive use of what is considered a STEAM approach to engineering education [1], an approach we describe in detail later in the paper. Over the last decade, we have seen how a range of employers, especially from the creative industries, are keen to employ graduates from our courses. Anecdotally, the graduates demonstrate a more creative and experimental approach to engineering, and are able to adapt quickly to using new technologies in new settings.

But anecdotal evidence is not enough. We would like to know the measurable impact, if any, this pedagogical approach actually has on student learning. Do students

learn to code in a different way with this approach, compared to more traditional approaches? If they do, does this impact positively upon learning outcomes? In this paper, we set out to investigate these questions, as we find there is a lack of research explicitly comparing how different pedagogical approaches impact on the way students learn programming.

Our first step towards addressing this apparent gap in knowledge has been to design and deploy a novel technological platform for teaching programming. The system has been designed to support our teaching and research by providing specific interactive, data gathering and analytics functionality. The system includes a browser based, integrated development environment, which is accessible online[1]. It has been used by hundreds of students in our department and its data gathering functionality makes it possible to conduct detailed, quantitative analyses of student coding activity. We provide full technical details of the system in [31], and highlight the relevant features for this study in section 3.1.

In this paper, we use this system to examine and compare the activities of students when they are exposed to STEAM and non-STEAM style computing lessons. This allows us to produce clear evidence about how people change their coding styles depending on the type of instruction they are given.

1.1 The need for inclusive coding instruction

Another aspect of our endeavour to understand how people respond to different instructional styles when learning programming is the need to make coding instruction more inclusive. Coding has been highlighted by industry leaders as 'the red thread that runs through Europe's future professions' [2]. To maximise the number of 'computational thinkers' [3] 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" [4].

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 approaches actually impact on the way a student goes about programming and, if so, is one approach better than

1 https://live.codecircle.com

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.

As we will demonstrate in the following report, 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 discuss background work around STEAM pedagogy and approaches to analysing coding activity. In section 3 we describe the experimental and technological framework we have developed to enable studies into computer science education. In section 4 we will describe the method used for this particular study. In section 5 we will describe and analyse the data that resulted from the study. In section 6 we will discuss and evaluate the results, concluding in section 7.

2 Background

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

2.1 STEAM and non-STEAM

STEAM is an approach to teaching engineering, science, technology and maths which adopts methods from art school teaching to better teach "problem solving, fearlessness, and critical thinking and making skills" [5]. Students are encouraged to construct their own ontologies of understanding through an active process of creation [6] Rose and Smith stated: 'the STEAM agenda should be about deep, sustained, powerful engagement as a way of learning' [7] The ideas and methods in STEAM are not new, though. It is based on a constructivist theory of learning, after Piaget, and is influenced by Dewey, a key figure in arts pedagogy who believed in teaching the whole person and that "inquiry [was] of necessity an experimental transaction" [8]. Thus STEAM can be seen as the latest in a series of related approaches, which we can trace back to Dewey's experience driven education [**Error! Reference source not found.**], Papert's constructionism [10] and Rutherford's inquiry based learning [11].

Science educators have adopted many of the techniques underpinning STEAM, for example, inquiry learning and project based learning [12], but this type of instruction has been attacked by some educational psychologists. Kirschner et al. claimed that 'unguided learning' stands to fail as a pedagogical approach since it involves an excessive cognitive load that will interfere with the basic mechanisms of learning, namely the interaction between working memory and long term memory [13]. Minner et al. responded to this work by providing a meta study, through which they were able to dismiss Kirschner et al.'s basic description of these learning methods as 'not the way that most inquiry-oriented practitioners or researchers would describe these kinds of instructional approaches' [14]. Hmelo-silver et al. provided further clarity about this by carrying out another meta study, this time examining the use of scaffolding in inquiry based learning [15]. They explain that scaffolding is used exactly for the purpose of reducing cognitive load during inquiry learning, thus addressing Kirschner et al's key issue about the excessive cognitive load caused by this type of instruction. They also highlighted the impact of this type of instruction on the more holistic goals of education, such as soft skill development, which takes us back to Dewey, who emphasised the importance of teaching the whole person in their social context.

Surprisingly, given its practical and applied character, this style of instruction has been slow to catch on in engineering education, as noted by Ben-Ari [16]. Many educators still employ 'chalk and talk', where the sage on the stage transmits knowledge to the receptive vessels in front of them, which can then be measured through the trusted and rigourous method of written examination. Despite its persistence, there is very strong evidence against the efficacy of this approach. In the largest meta-study to date of the impact of active, constructivist learning in STEM education, Freeman et al. stated that failure rates under traditional lecturing increase by 55% over the rates observed under active learning and that average examination scores after active learning improved by about 6% [17]. Given these results and the fact that STEAM is a quite extreme form of active learning, we consider it a very interesting pedagogical approach to investigate.

This discussion of well established education theory is necessary as a background to our work, as there has been a lack of work in the computing education literature that builds on previous theories, as noted by Malmi et al [18]. Therefore, we have designed the study reported here based on a clear 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 [19].

2.2 Analysing coding behaviour

We shall now consider some examples of work that describes and analyses student programming behaviour, since this is the key mode of analysis in this paper and this work has inspired some of the approaches we use. Rodrigo and Baker looked at programming behaviour, such as repeated attempts to compile the same code [20]. They developed a model that was able to detect affects such as frustration in programming

labs. 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 [21]. The techniques included observing the sizes of changes in the programs over time using abstract syntax trees. Using these metrics, they clustered and classified learners, mapping their student classifications onto Papert's tinkerer and planner categories [10]. Our work differs in that we are looking at a higher time resolution, we do not analyse the programs themselves, and we focus on comparing pedagogical approaches, not comparing students.

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 [22]. They reported significant increases in self efficacy regarding coding in students using the system [23]. 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, as opposed to investigating the high level impact of STEAM on student self-efficacy.

Fields et al. examined the use of initialisation, events and parallelism in Scratch programs written by youths at a summer school [24]. They used a constructionist pedagagy and were able to make observations about different students and their development of these programming concepts. Our study differs from this one in that we are looking at lower level information about the coding behaviour, and we actively compare different pedagogies.

Yang et al. developed a method to model and analyse informal learning in online communities of Scratch programmers [25]. They modelled the amount of learning, the speed of learning and the amount of prior knowledge, all based on patterns of use of the 170 different blocks available in Scratch (the vocabulary). They clustered learners based on their trajectories through a weighted vocabulary space. Our work differs in that we have a formal learning context, we are not considering programming vocabulary, and we compare different teaching techniques as opposed to comparing different learners.

As we can see, there is much interesting work analysing student programs and programming styles, but what is perhaps lacking is a comparison of how programming behaviour changes in response to different pedagogical approaches. Our work addresses this gap in the literature, with a comparison of STEAM and non-STEAM approaches.

3 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.

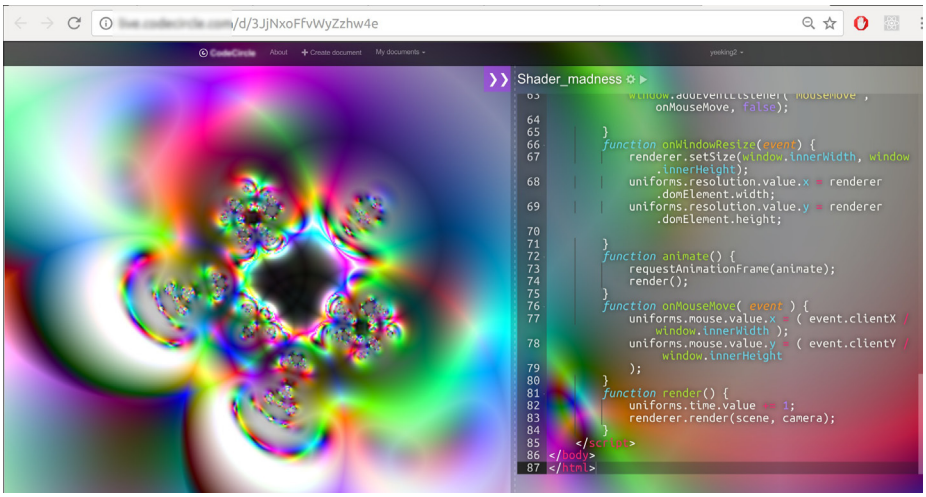

Fig. 1. Screen shot of the IDE. The code for the program can be edited on the right. The output of the running program can be seen on the left. The code editing panel is made opaque when editing so the code can be clearly seen

3.1 The educational IDE

We have developed an educational, browser based IDE which allows students to write programs in their web browser. 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 [26].

The essential feature from the above list for the experimental framework is the ability to gather high resolution, code editing log data. This data 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 underlying 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 [27]. 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 single students were making during the lessons.

3.2 Learning analytics

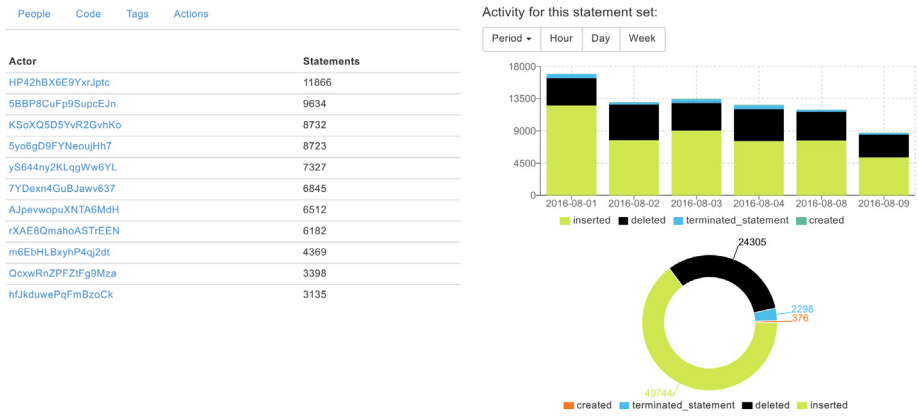

Fig. 2. Screen shot of the learning analytics dashboard. On the left is the data browser, currently showing an anonymised list of users; on the right are two visualisations of coding behaviour over the 6 days of the study.

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 [28]. Briefly, xAPI provides a formal way of describing interactions between learners and learning technology as a set of xAPI statements in JSON format. It is common for researchers to develop xAPI 'recipes' for particular learning activities, and we have defined recipes suitable for describing the activities of programmers, and published it as a github repository[2] as we found that there were no pre-existing recipes for such. The main elements of a generic xAPI 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 'termi-

2 https://github.com/yeeking/xapi-coding-recipes

nate' (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.

3.3 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 [29] - 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, and the data gathering should be non-instrusive as much as possible.

4 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, two hour, group lessons over two weeks, wherein six activities were STEAM and six 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 two weeks prior to the study 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 structure of the learning activities. The programming activities in the lessons fell into four 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 constrained problem requiring 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 to explore 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. It also required that they operate within a community of their peers, sharing and co-creating.

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.

5 Results and analysis

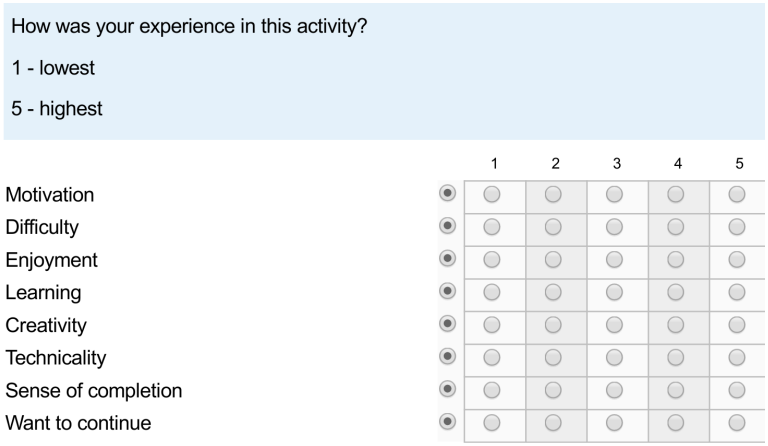

Fig. 3. Screenshot of how the survey looked in our VLE

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 conditions between which we can compare student experience and coding behaviour. In the following subsections, we will describe four analyses of the data: 1) self reported experience metrics, 2) activity levels in the lessons, 3) activity patterns in the lessons and 4) the relationship between final grades and activity patterns.

5.1 Self reported experience

Table 1. 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

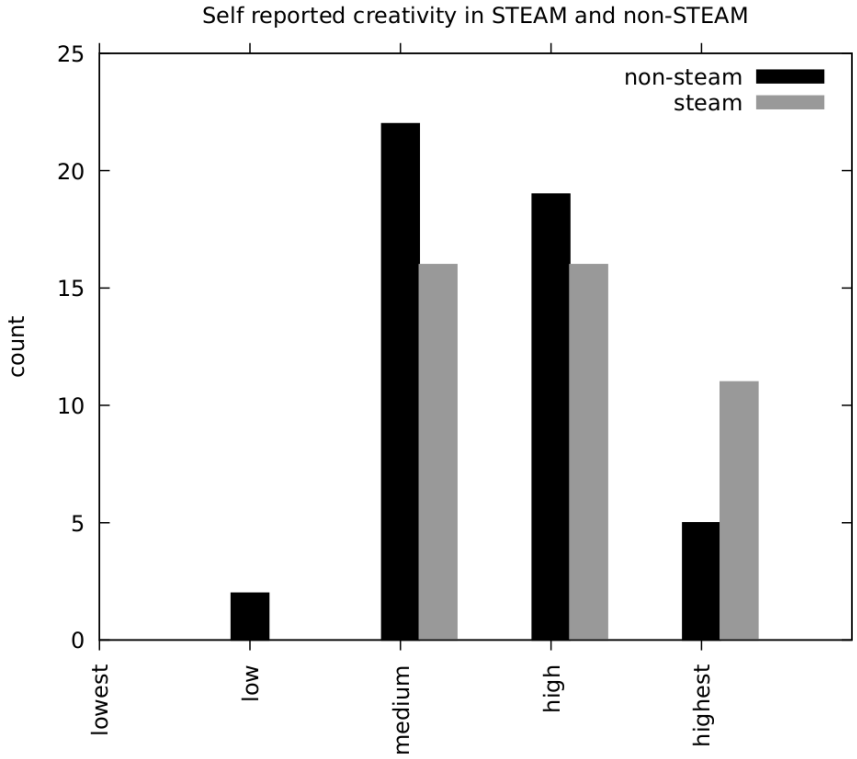

Fig. 4. Histogram of the two distributions of self reported creativity, showing the tendency for students to report higher levels of creativity in STEAM

In our first analysis, we measured the significances of the variation between the answers to the post STEAM and non-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 1**Error! Reference source not found.**. 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 from the two lesson types 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.

5.2 Coding activity

In our second analysis, we looked at the number of editing actions made in the different lessons. Table 2 shows the total, mean and standard deviation of each type of action, separated by lesson type. The total number of actions taken in the non-STEAM lessons is greater, and the difference seems to be made up by extra insert actions. There are more insert actions in non-STEAM lessons by approximately one standard deviation of the non-STEAM lesson set. We will discuss this result in the discussion section.

Table 2. Actions of each type, logged in STEAM (S) and non-STEAM lessons (NS) by the 11 participants

Lesson	create	delete	insert	terminate
S all	190.0	12605.0	20551.0	1013.0
NS all	186.0	11700.0	29193.0	1285.0
S mean	31.7	2100.8	**3425.2***	168.8
NS mean	31.0	1950.0	**4865.5***	214.2
S std	6.1	412.8	712.6	76.3
NS std	7.0	655.3	1387.6	75.2

Figures 5 and 6 show the variation of code editing activity levels during the 2 hour lessons for all students. The aim is to show if students were active throughout the lessons or if the activity level varied. A smooth gradient indicates steady activitty, a jagged gradient indicates students stopping and starting. The non-STEAM lessons appear to have a slightly smoother activity curve, indicating more consistent activity levels during the lessons. We will discuss these graphs further in the discussion section.

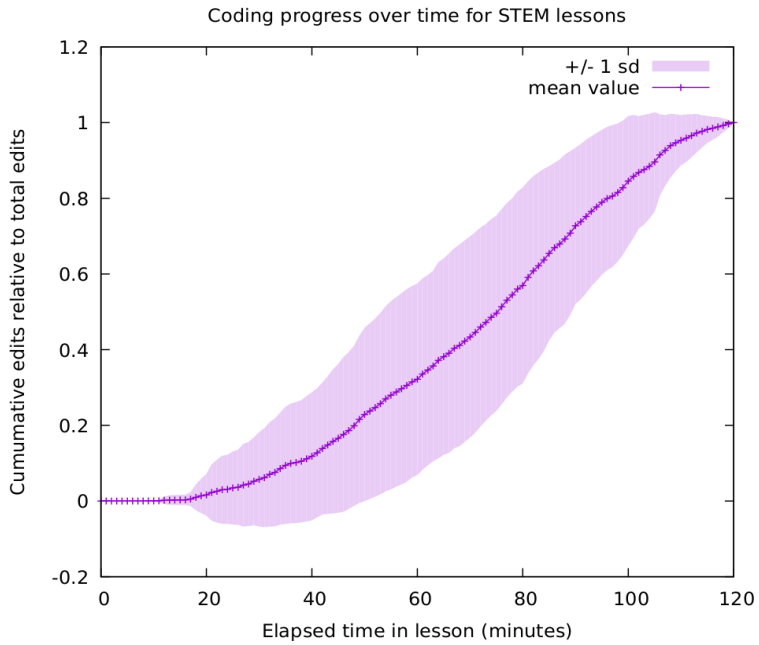

Fig. 5. Cumulative edits made in the lessons against time for all students in non-STEAM lessons. The number of edits is normalised against the total number of edits.

Fig. 6. Cumulative edits made in the lessons against time for all students in STEAM lessons. The number of edits is normalised against the total number of edits.

5.3 Coding activity ratios

In our next analysis, we investigated the relationship between insert and delete actions, which the previous analysis suggested was a key differentiator between the two lesson types. 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 8 where each box represents a verb, and its size and position indicate the range of values observed for that verb's occurrence ratio. 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. Or, relatively fewer inserts. 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

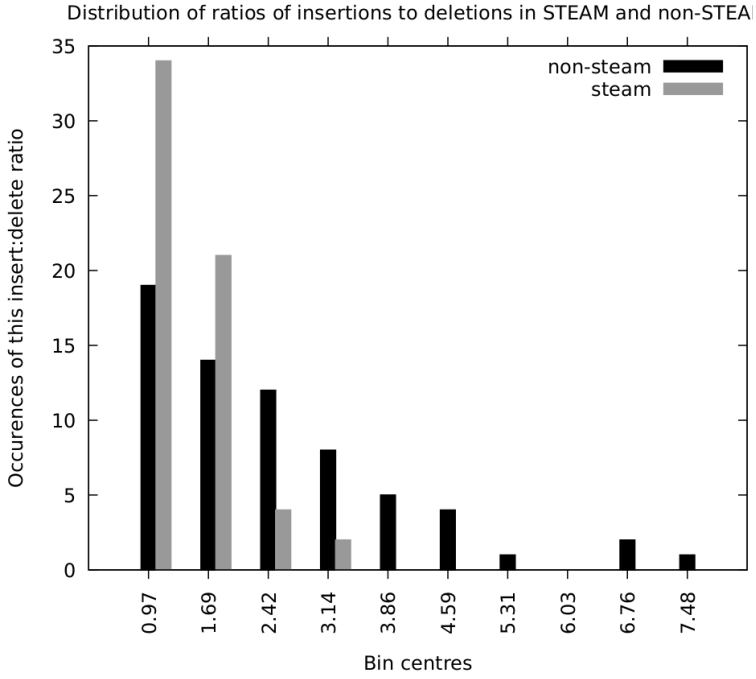

Fig. 7. Distribution of insert:delete ratios in STEAM and non-STEAM. A value of 1 means the same number of deletions and insertions happened in a lesson. A value > 1 means there were more insertions than deletions.

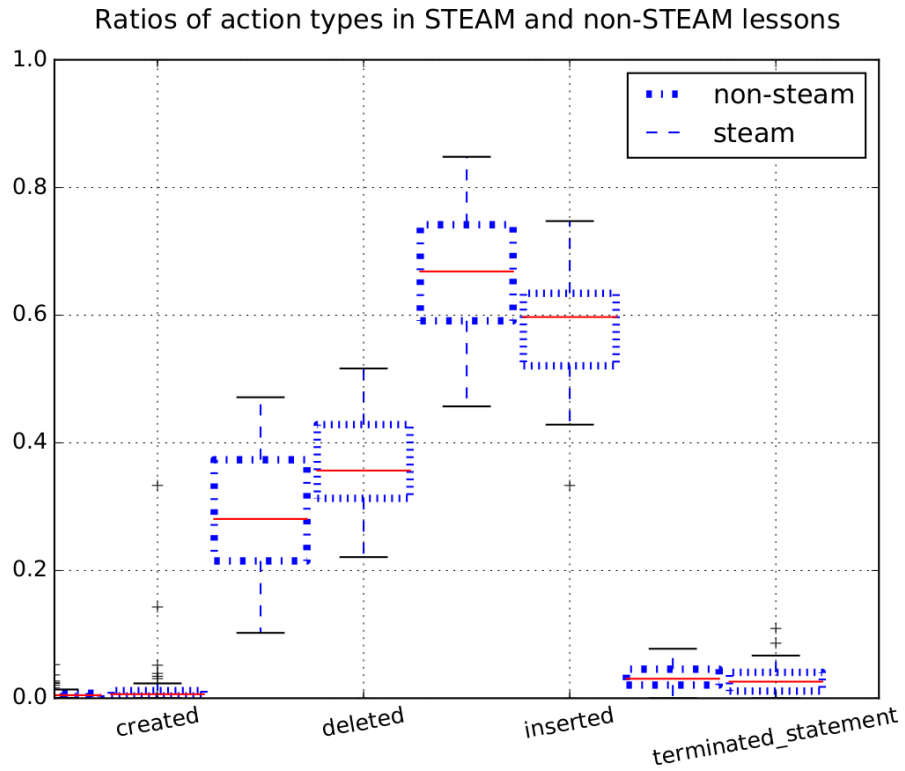

Fig. 8. Ratios of code operation types in STEAM and non-STEAM lessons. Each box shows the range of values for a verb in a lesson type. High up boxes indicate more common actions, box size indicates the range of values observed

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 7**Error! Reference source not found.**. 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).

5.4 Correlating final grade with activity ratios

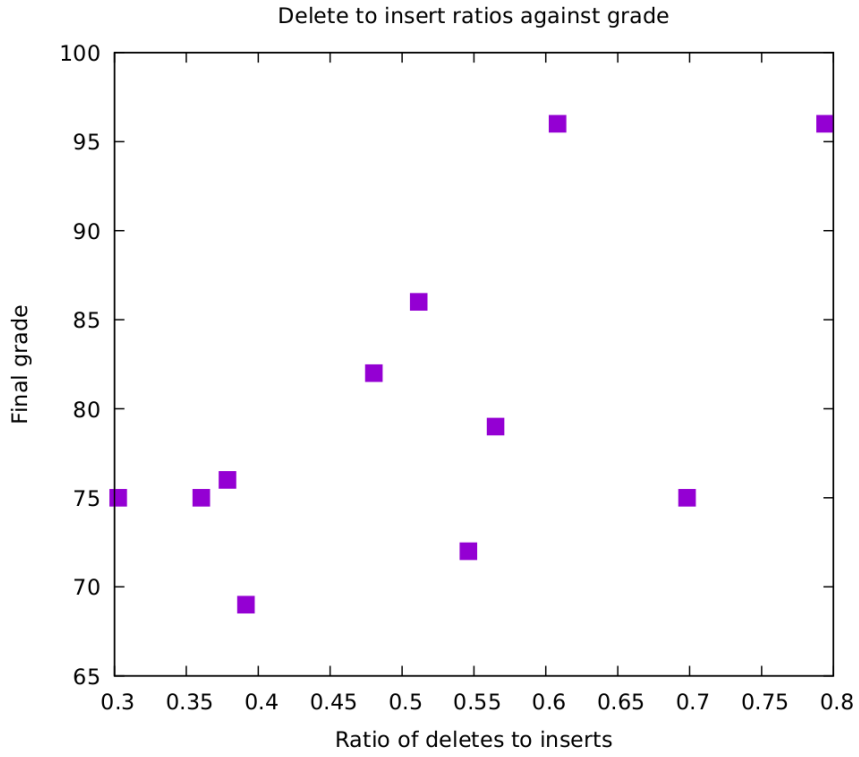

Fig. 9. Final grades achieved by students plotted against their ratio of deletion to insertion edits across all lessons. A value of 0.5 on the axis means there were twice as many insertions as deletions, a value of 1 on the x axis means there were the same number of insertions and deletions.

In the next analysis, we follow up on the observation of greater relative deletion behaviour in STEAM lessons and ask the question: is deleting code something that successful students tend to do more relative to inserting code? 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 students 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 9. 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. In conclusion, there are two clear, final grade related features of this relatively small dataset: 1) successful students do not tend to generate more actions overall in lessons and 2) successful students tended to have more deletes relative to inserts.

6 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.

6.1 Do students code differently when undertaking STEAM and non-STEAM exercises?

We have defined four lesson types - two are STEAM and two are non-STEAM. 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 in several ways. The total number of coding operations observed in the non-STEAM lessons was greater. In particular, the number of code insert operations was greater. Secondly, 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, or less inserts 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.

6.2 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 five 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.

6.3 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 need to carry out a longer term study to gain a greater understanding of this suggestion.

7 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 our institution, and the lack of programming-specific, comparative studies in the literature. 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 other areas of their 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 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. This provides the perfect opportunity to expand the study. 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 [30].

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

8 Acknowledgments

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.

9 References

[1] 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, 2014. https://doi.org/10.1109/TALE.2014.7062556

[2] 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 =0pt

[3] J. M. Wing, "Computational Thinking," *Communications of the ACM*, vol. 49, no. 3, pp. 33–35, 2006. https://doi.org/10.1145/1118178.1118215

[4] Margolis and J. Goode, "Ten Lessons for Computer Science for All," *ACM Inroads*, vol. 7, no. 4, pp. 52–56, nov 2016. [Online]. Available: http://doi.acm.org/10.1145/2988236 =0pt

[5] J. Maeda, "STEM + Art = STEAM," *The STEAM Journal*, vol. 1, no. 1, pp. 1–3, 2013. [Online]. Available: http://scholarship.claremont.edu/steam/vol1/iss1/34 =0pt

[6] 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 =0pt

[7] C. Rose and B. K. Smith, "Bridging STEM to STEAM: Developing new frameworks for Art-Science-Design Pedagogy," *Rhode Island School District Press Release*, 2011.

[8] J. Dewey, A. W. Moore, H. C. Brown, G. H. Mead, B. H. Bode, H. W. Stuart, J. H. Tufts, and H. M. Kallen, *Creative intelligence: Essays in the pragmatic attitude*. plus 0.5em minus 0.4emH. Holt, 1917.

[9] J. Dewey, *Experience and education*. plus 0.5em minus 0.4emNew York: McMillan, 1938.

[10] S. Papert, *Mindstorms: children, computers, and powerful ideas*. 1em plus 0.5em minus 0.4emBasic Books, Inc. New York, NY, USA, 1980. [Online]. Available: http://dl.acm.org/citation.cfm?id=1095592 =0pt

[11] F. J. Rutherford, "The role of inquiry in science teaching," *Journal of Research in Sci ence Teaching*, vol. 2, no. 2, pp. 80–84, 1964. https://doi.org/10.1002/tea.3660020204

[12] J. E. Mills, D. F. Treagust, and R. M. B. Ãž, "Engineering Education - is Problem- based or Project-based learning the answer?" *Autralasian Journal of Engineering Education*, vol.

3, no. 2, pp. 2–16, 2003. [Online]. Available: http://www.aaee.com.au/journal/2003/mills_treagust03.pdf =0pt

[13] P. A. Kirschner, J. Sweller, and R. E. Clark, "Why Minimal Guidance During Instruction Does Not Work," *Educational Psychologist*, vol. 41, no. March 2015, pp. 87–98, 2006. [Online]. Available: http://www.cogtech.usc.edu/publications/kirschner_Sweller_Clark.pdf =0pt

[14] D. D. Minner, A. J. Levy, and J. Century, "Inquiry-based science instruction-what is it and does it matter? Results from a research synthesis years 1984 to 2002," *Journal of Research in Science Teaching*, vol. 47, no. 4, pp. 474–496, 2010. https://doi.org/10.1002/tea.20347

[15] C. E. Hmelo-silver, R. G. Duncan, and C. A. Chinn, "Scaffolding and Achievement in Problem-Based and Inquiry Learning : A Response to Kirschner , Sweller , and Clark (2006)," *Educational Psychology*, vol. 42, no. 2, pp. 99–107, 2007. https://doi.org/10.1080/00461520701263368

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

[17] S. Freeman, S. L. Eddy, M. McDonough, M. K. Smith, N. Okoroafor, H. Jordt, and M. P. Wenderoth, "Active learning increases student performance in science, engineering, and mathematics," *Proceedings of the National Academy of Sciences*, vol. 111, no. 23, pp. 8410–8415, 2014. https://doi.org/10.1073/pnas.1319030111

[18] 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.

[19] 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.

[20] 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, 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1584322.1584332 =0pt

[21] 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*, vol. 23, no. 4, pp. 561–599, 2014. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/10508406.2014.954750 =0pt

[22] 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.

[23] B. Magerko, J. Freeman, T. O. M. Mcklin, S. Consulting, G. Llc, M. Reilly, E. Livingston, S. Mccoid, B. Magerko, J. Freeman, T. Mcklin, M. Reilly, E. Livingston, and S. Mccoid, "EarSketch: A STEAM-Based Approach for Underrepresented Populations in High School Computer Science Education," *ACM Transactions on Computing Education*, vol. 16, no. 4, 2016.

[24] D. A. Fields, O. M. Hill, J. Amely, and J. Maughan, "Combining Big Data and Thick Data Analyses for Understanding Youth Learning Trajectories in a Summer Coding Camp," in *SIGCSE '16 Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, 2016, pp. 150–155. https://doi.org/10.1145/2839509.2844631

[25] S. Yang, C. Domeniconi, M. Revelle, M. Sweeney, B. U. Gelman, C. Beckley, and A. Johri, "Uncovering Trajectories of Informal Learning in Large Online Communities Of

Creators," in *L@S '15 Proceedings of the Second (2015) ACM Conference on Learning @ Scale*, 2015. https://doi.org/10.1145/2724660.2724674

[26] 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.

[27] J. Gentle, "ShareJS - Live Concurrent Editing in your App," 2012. [Online]. Available: https://github.com/josephg/ShareJS =0pt

[28] ADLNET, "xAPI-Spec," 2016. [Online]. Available: https://github.com/adlnet/xAPI-Spec/blob/master/xAPI.md =0pt

[29] 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. https://doi.org/10.1207/s15326985ep3904_1

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

[31] 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

10 Authors

Matthew Yee-King is a Lecturer in the Department of Computing at Goldsmiths, University of London. His research interests include analytics and evidence driven learning technology, computer science education, and machine learning applied to signal processing.

Mick Grierson is a Reader in the Department of Computing at Goldsmiths, University of London. His research interests are in applied Computer Science and Engineering for the Creative Industries, specifically digital signal processing, machine learning and interaction.

Mark d'Inverno is Professor of Computer Science at Goldsmiths, University of London. He has a range of experience leading interdisciplinary research projects across music, education, art and design.

This article is a revised version of a paper presented at the EDUCON2017 conference held in Athens, Greece, 25-28 April 2017. Article submitted 04 July 2017. Published as resubmitted by the authors 17 August 2017.