Effect of a 2-week Scratch Intervention in CS1 on Learners with Varying Prior Knowledge

Shitanshu Mishra
IDP in Educational Technology
Indian Institute of Technology Bombay
Mumbai, India
shitanshu@iitb.ac.in

Sudeesh Balan
IDP in Educational Technology
Indian Institute of Technology Bombay
Mumbai, India
sudbalan@gmail.com

Sridhar Iyer
Department of CSE
Indian Institute of Technology Bombay
Mumbai, India
sri@iitb.ac.in

Sahana Murthy
IDP in Educational Technology
Indian Institute of Technology Bombay
Mumbai, India
sahanamurthy@iitb.ac.in

ABSTRACT
A large CS1 class often needs to provide scaffolding for novices while keeping advanced learners engaged. Scratch has been shown to be suitable to address a diverse set of requirements. In this study, we determine whether a 2-week Scratch intervention in a CS1 course is useful from two perspectives: i) as a scaffold for novices to learn basic programming concepts and transition to C++, and ii) as a tool for advanced learners to remain engaged and do challenging work. We conducted a field study of 332 first-year undergraduate engineering students, two-thirds of whom were novices. We analyzed student performance on exams and Scratch projects. We administered a survey to determine student perceptions on the usefulness of Scratch. Some key findings of our study are: (i) Novices were able to catch-up to advanced learners in Scratch questions of the type 'Predict the output' and 'Debug the program', (ii) Projects by advanced learners reached 80% of the complexity of 'most loved projects' on the Scratch website, and (iii) 69% of students perceived Scratch to be useful for learning programming concepts and transitioning to C++.

Categories and Subject Descriptors
K.3.2 Computer Science Education.

Keywords
CS1; scratch intervention; novice learners; advance learners;

1. INTRODUCTION
In typical Indian universities, there is a single programming course for all freshmen engineering students, with the main programming language taught in the course being C++ or Java. Students come with widely varied programming experience, ranging from those with zero prior exposure, to those already competing in programming contests. Thus, novices get daunted because they have to simultaneously learn both new constructs and syntax, while advanced learners tend to get bored when basic concepts are being taught. Instructors face a dual challenge: (i) quickly equip novices with basic programming concepts and skills so that they can keep up with the main learning outcomes of the course, (ii) keep the advanced learners engaged. Hence we need a solution that addresses both these problems. In this study, we examine the suitability of a 2-week Scratch curriculum as a solution for both novices and advanced learners, in a large CS1 course. Scratch has been found to facilitate the learning of programming and computer science concepts [11]. It has been shown to have: a "low floor" (easy to get started with); "high ceiling" (complex projects can be built); and "wide walls" (can address variety of topics and themes) [13], especially at the middle school level, with some attempts to use it in a CS1 course at the college-level [20]. It has been recommended for novice programmers for its ease of use, as well as for advanced programmers for its facility to build complex projects [9].

Motivated by the potential of Scratch to address such diverse needs, the research goal of this study is to investigate the impact of a short intervention of Scratch in a large, academically diverse, college-level CS1 course on C++ programming. Most Scratch studies (except 2 or 3 such as Malan's [20]) are done in middle school while ours is at the college level. We determine the effectiveness of Scratch from two perspectives: i) as a scaffold for novices to learn basic programming concepts and transition to C++, and ii) as a tool for advanced learners to remain engaged with content. Our research questions (RQ) are:

RQ1: How much have novices learnt, in terms of: (a) basic programming concepts and (b) how much were they able to transition to C++?
RQ2. What are the benefits of Scratch to advanced learners?
RQ3: How useful do students perceive Scratch to their learning of programming concepts and their engagement?

We conducted a field study of implementing Scratch in a large CS1 class to answer the above research questions. Our treatment consisted of two weeks of Scratch instruction that included four lectures, three labs and a project. Our research was conducted in-situ, instead of controlled lab experiments, as we were interested in determining the effects of deploying Scratch as a short-term intervention in a regular university classroom setting, wherein the rest of the course was taught using C++. We used a mixed-methods research design to answer the research questions (RQs). We categorized students as novices or advanced learners based on their responses to a questionnaire on prior

ITICSE'14, June 21–25, 2014, Uppsala, Sweden.

http://dx.doi.org/10.1145/2591708.2591733

programming background, which was administered in the first week of the semester. Students' learning of programming concepts after instruction was assessed via questions on an in-semester quiz and mid-term exam (RQ1). Data from one other source was used to determine the extent of novice students' application of programming concepts and skills (RQ1). This source was a non-traditional assessment tool in which students were asked to generate questions in different CS1 topics. Another data source was Scratch projects done by students. These were analyzed via Scrape tool [6], to determine the complexity of projects created by advanced learners, which we used to infer the engagement of advanced learners (RQ2). Finally, we administered a survey questionnaire to determine students' perceptions of the usefulness of Scratch for their learning and engagement (RQ3).

Our results showed that novices were able to catch-up to advanced learners in Scratch questions of type – 'Predict the output' and 'Debug'. However, we found that novices did not perform comparably on questions where they had to write a program of a numerical nature (such as generating Fibonacci sequence or Pascal's triangle). In case of advanced topics that can be easily taught via Scratch - threads and graphics - both novices and advanced learners showed evidence of learning. Projects by advanced learners reached 83% of the complexity of the "Most-loved projects" on the Scratch website [15] indicating high engagement. Student perceptions from the survey confirmed that Scratch was beneficial in helping them learn basic programming concepts and transitioning to C++, with 69% agreement. The most frequently cited benefit of Scratch was that it helped students overcome the barrier to start programming, which is a key insight regarding the usefulness of Scratch for students' affective component. The survey also showed that Scratch projects helped keep students engaged with the content.

2. THEORY AND RELATED WORK

In this section, we first discuss the theoretical basis for using Scratch (Section 2.1), from the perspective of reducing cognitive load and scaffolding for novices, and balancing skill versus challenge for advanced learners. We then focus on prior work on the use of Scratch in college-level courses (Section 2.2).

2.1 Theoretical basis for using Scratch

Students without any prior exposure to programming need to simultaneously learn the techniques to solve a problem, and the use of a programming language as a tool to solve the problem. These multiple demands on working memory could lead to cognitive overload [19] for a novice programmer. This prevents novice students from learning computer science concepts and has been reported to be a major problem in CS education [5]. One approach to reducing cognitive load has been the use of visual programming environments [1] that allow students to assemble code snippets using a drag-and-drop interface. Scratch is one language having such an environment.

A Scratch intervention can thus be considered as a scaffold for novice students. Scaffolding is a temporary support provided by an instructor to assist learners for dual purposes [12]: i) in completing a task that they otherwise would not be able to accomplish, and, ii) in enabling students to learn from that experience so that they are prepared to perform similar tasks in the future. Our Scratch intervention makes use of both functions of scaffolding by helping novice programmers to conceptually solve a problem without having to focus on the details of syntax, and to apply these programming concepts when they later transition to writing programs in C++.

Advanced learners, on the other hand, need a sufficiently challenging set of learning materials so that they remain engaged with the content. From the perspective of Flow Theory [4], students become and remain engaged in learning when there is equilibrium between the challenge level of the activity and the learner's personal skill. A high-school study has reported students to have experienced higher engagement when the perceived level of challenge and their skills were high and in balance with each other [17]. We exploit the 'high ceiling' feature of Scratch [13] to provide advanced learners with activities commensurate with their skill level.

2.2 Scratch in CS0/CS1

Scratch is a visual programming environment that provides various 'blocks' of programming constructs, such as statements, conditions, loops, operators, variables, and so on. These can be dragged and dropped onto a 'stage' area in 'stacks' that can be then executed. A detailed account of the features and capabilities of Scratch can be found in [8] and [13].

Scratch was designed to promote 'technological fluency' [13] among young learners. It enables users to create media rich 'projects' (Scratch programs) such as animation, music videos and games. In terms of programming, Scratch has been shown to be useful to learn concepts such as loops and conditionals [8].

Though majority of Scratch users are at the middle and high-school levels, Scratch has been used in a few formal courses at the college level [20]. In [9], Scratch was used in a CS1 course for a short period (1-2 weeks) before transitioning to more advanced programming constructs in Java. Students were first introduced to basic programming constructs using Scratch. They created Scratch projects with multiple stacks as part of the assignments. The study reported overall positive findings of student perceptions. In a different implementation for at-risk students [14], Scratch was deployed in a semester-long CS0 course to improve the retention and performance of students. Survey and performance data indicated that the CS0-Scratch course was effective in preparing students for the next level CS1 course. Our study includes both learning data and perception data.

We found that there are several other programming tools and techniques to help novices learn programming [16]. Some are visual programming environments, such as Alice [3] and JPie [18]. Others allow students to visualize the control flow of a program [7] and create mental models [10]. However, some researchers have commented that these tools either have a high learning curve for first-time programmers, or are restrictive as compared to Scratch [9]. Hence we did not explore the use of these tools for our intervention.

3. COURSE IMPLEMENTATION

The setting for our study was a large enrollment CS1 class of 450 first year undergraduate students, across various engineering disciplines, excluding CS majors; the student characteristics are described in Section 4.1. The goal of the CS1 course was to teach programming concepts and C++ skills. The course was conducted over 14 weeks in Spring 2013, using lectures and labs. Our Scratch intervention was limited to the first two weeks of lecture and three weeks of lab. The details are as follows:

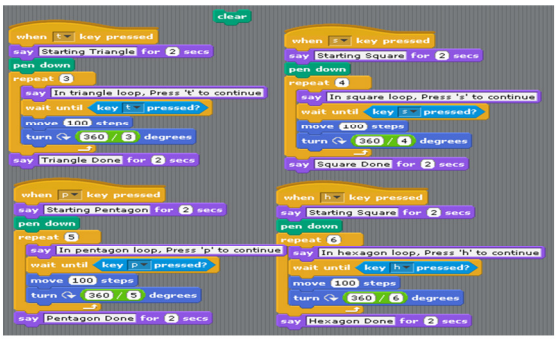

Figure 1: Example program -'predict the output' activity

- *Lecture*: The blocks on motion, control, sensing, operators, and variables, were discussed in detail, with lesser emphasis on the blocks on looks, sounds and pen. Additional attention was given to the advanced constructs such as lists (arrays) and event handling (broadcast-receive). The lectures had an active learning structure. For example, the program shown in Figure 1 was used in the second week. Students were required to make predictions to answer questions like "What will happen if keys *s* and *t* are pressed simultaneously?"
- *Lab*: The lab activities depended on the lecture contents of the previous week. The students worked in pairs. The first lab got students familiar with the Scratch environment, by getting them to run and modify a given program. The second lab required them to play a game of space-invaders, already implemented in Scratch, do a code walk-through, and then modify the code for altering a given behavior of the game. The third lab had students use the single step mode to observe threads interleaving, debug programs, and also work on their projects as described below. Activities from the fourth lab onwards were using C++.
- *Project*: In parallel with labs two to four, the student pairs also worked on building a game for their Scratch project, which they demonstrated to the TA at the start of lab five. A description of the projects is given in Section 4.2.2 and their analysis is presented in Section 5.2.
- *Question Generation*: In lab five, each student pair was asked to generate two questions that other students could practice on. A description of this activity is in Section 4.2.3 and the corresponding analysis is presented in Section 5.1.
- *Assessment*: One quiz (1-hour written exam) and one midterm had Scratch questions, to test students' conceptual understanding along traditional lines. Midterm had C++ questions also, which were used to address RQ1b.

This intervention meets our requirements as: 1) Scaffolding for novices was achieved by the gradual ramp-up of the concepts in the two weeks of lecture, followed by applying the concepts in the labs. 2) Engagement for advanced learners was achieved by having the project start in the second week, and giving students the freedom to decide the nature and complexity of their project. The only incentive for performance was that the top few projects were to be showcased as the 'Hall of Fame'.

4. STUDY METHODOLOGY

We used a mixed methods design and triangulated our data with multiple types of measurements for each research question. We describe the data collection instruments and the analyses in Section 4.2. Now, we first describe our sample and how we categorized students into 'novices' and 'advanced learners'.

4.1 Sample

There were 450 students registered for the class (395 male, 55 female). All were first year students majoring in different branches of engineering. Students who were admitted to our institute were among the highest ranked in an extremely competitive exam testing analytical skills in mathematics, physics and chemistry (they were all among the top 1000 out of 500000 students). Hence all students in the study can be considered as equivalent in all respects, except prior exposure to programming.

To determine prior programming background, we conducted a survey in the first week of the semester. Students had to respond to a series of questions on their familiarity with computers and programming experience. Those who had opted for programming electives in Grades 10 or 12, or had done programming outside of school, were classified as 'advanced learners'. Others were classified as 'novices.' 332 students completed the above survey, hence only those students were considered in the sample. We found 217 students were novices, and 115 were advanced learners. 10 advanced learners had participated in programming contests.

4.2 Data collection tools and analysis

4.2.1. Quiz and mid-term exam question scores

Scores from the quiz (4th week) and the mid-term (6th week) were used to determine students' acquisition of basic programming concepts (RQ1). The quiz contained four questions, all of which were based on Scratch, while the mid-term exam contained three questions, of which one was based on Scratch and the others on C++. There was a mix of conceptual and programming question types, including: 'predict the output', 'debug the program', as well as 'write a program'. These were typical CS1 questions. For example, a program for BubbleSort with erroneous array initialization and loop conditions was given and students were required to debug the program. In another question, they were required to write a program to output the Fibonacci series. We did not analyze scores from tests after the mid-term exam since the topics in the latter half of the semester were not related to the Scratch intervention.

Analysis

We analyzed novice learners' performance on the above questions and examined the scores in each category of questions. We have also used scores of advanced learners as a benchmark to estimate the absolute learning gain for Novices (RQ1).

4.2.2 Scratch projects

At the end of the second week, students were asked to work on a Scratch project in teams of two. They were expected to exercise their creativity and also demonstrate their learning, by writing games using multiple scripts and blocks. Students worked on their projects for a period of two weeks during their labs but were free to work outside of lab hours also. Projects were graded by lab TAs and exceptionally good projects (with large number of sprites, having complex interactions and emulating real-world functionality) were hosted on the course website as 'Hall of Fame' entries.

Analysis

We analyzed the Scratch projects using the Scrape visualization tool [6]. The Scrape tool provides a record of the programming constructs used in each project, such as, variables, sprites, stacks and blocks. We analyzed the frequency of use of different constructs by advanced learners and students whose projects were showcased in the Hall of Fame. We used the reference of

Table 1: Scores on exam questions of different types in Scratch and C++ (Maximum Marks for each question: 10)

	Novice Average (SD)			**Advanced** Average (SD)		
	Predict output	**Debug a program**	**Write a program**	**Predict output**	**Debug a program**	**Write a program**
Scratch	Ques.1 - 7.5 (3.4) Ques.2 - 7.2 (2.2)	Ques.3 - 6.9 (3.5) Ques.4 - 4.6 (4.5)	Ques.5- 3.4 (3.3) Ques.6 - 2.8 (3.5)	Ques.1 - 8.2 (3.0) Ques.2 - 7.6 (1.9)	Ques.3 - 8.1 (3.0) Ques.4 - 5.4 (4.2)	Ques.5 - 6.8 (3.2) Ques.6 - 5.8 (3.9)
C++	NA	Ques.7 - 7.1 (4.3) Ques.8 - 5.4 (3.7)	Ques.9 - 6.4 (2.9)	NA	Ques.7 - 9.2 (2.3) Ques.8 - 7.8 (2.8)	Ques.9 - 9.1 (1.9)

average frequency distribution of constructs used in the 'Most-loved' projects [15], as an estimate of 'high ceiling' [13], while examining the performance of our advanced learners (RQ2).

4.2.3. Question-generation by students

In the 5th lab, students did a question-generation exercise. Each pair of students was asked to generate two questions, pertaining to the topics covered so far, which could be given as practice questions for the next lab-batch. They were free to set either a programming problem or a conceptual question, and had to submit detailed answers to their generated questions. They were given only one open-ended guideline "The questions should be challenging but should not be too difficult for the students in the next batch to complete in the lab". We used the question-generation exercise as an independent measure to examine the extent of application of programming concepts, and hence of students' learning of the concepts (RQ1).

Analysis

We analyzed each question generated on the basis of the programming concepts targeted by it. These concepts were chosen from computational thinking concepts described in [2], and include: Sequence, Loops, Threads, Events, Conditionals, Operators, Variables, and Arrays.

4.2.4. Survey questionnaire

We created a survey to address RQ3: How useful do students perceive Scratch to their learning of programming concepts and their engagement? The survey had five questions, the first three of which were on a Strongly Agree to Strongly Disagree Likert scale. **Q1** was on whether students perceive Scratch useful for learning programming concepts; **Q2** was to determine students' perception of the usefulness of Scratch to transition to C++; **Q3** asked students if they enjoyed programming with Scratch; the last two questions were open-ended. In **Q4**, students were asked to write the main benefit they found in using Scratch, and in **Q5** they were asked for the most frustrating aspect in using Scratch.

Analysis

Quantitative analysis of first three questions gives the measure of students' engagement and their perception about usefulness of Scratch for their learning. Open ended - Questions 4 and 5 were analysed using content analysis technique to examine the advantages-disadvantages of Scratch as perceived by students.

5. RESULTS

Recall that our goals were to explore use of Scratch as : i) a scaffold for novices to learn basic programming concepts and transition to C++, and ii) a tool for advanced learners to remain engaged and do challenging work. Section 5.1 gives the results corresponding to goal 1 which relates to RQ1 and RQ3. Section 5.2 gives the results corresponding to goal 2 which corresponds to RQ2 and RQ3.

5.1 Learning of Programming concepts by Novices

5.1.1 Acquisition of programming concepts and transitioning to C++: Exam scores

Table 1 shows exam scores of novices and advanced students in different types of questions in Scratch and C++. An independent sample t-test shows that novices were able to catch-up to advanced learners in Scratch questions of type – 'Predict the output' and 'Debug'. Novices were not able to catch-up in C++ 'Debug' questions, perhaps because they did not get adequate time to get familiar with the syntax; the test was conducted within 3 weeks after the transition. Novices were not able to catch-up in any 'Write a program' question, which is somewhat to be expected (RQ1).

The correlation analysis between midterm scores in the Scratch with midterm scores in C++ indicated a moderate positive linear relationship. In addition to this, 65% of the students who did well (score higher than 1 SD above mean) in Scratch in the midterm exam, also did well in the C++.

5.1.2 Application of Programming Concepts: Question-Generation

Table 2 shows the number of student-pairs who generated questions involving various programming concepts. We labeled a pair as 'advanced' if at least one of them was an advanced learner as per 'prior programming background' survey. The presence of a concept in a question generated by students indicates that they have learnt and applied that concept.

Table 2: Concepts addressed in Question-generation activity

Concepts	**Questions generated by novices** (% questions in which this concept is addressed) total no. of questions =84 *Note: % questions generated by **advanced** learners are given in ( )*
Sequence	92 (90)
Loops	65 (83)
Threads	10 (5)
Events	11 (5)
Conditionals	61 (66)
Operators	94 (83)
Data	95 (93)
Arrays	10 (24)

We note that for concepts of sequence, data and conditionals, the percentage of questions generated by novices was comparable to that of advanced learners. Moreover, since all students submitted solutions to their generated questions, it shows that students are

not only addressing different programming concepts but are also comfortable in solving questions requiring these concepts.

5.1.3 Student Perception of Learning

We show a summary of responses to the first two Likert-scale items (Questions 1, 2) on the student perception survey. To simplify the presentation, we have combined responses in the Strongly Agree and Agree categories as Positive, and the Strongly Disagree and Disagree as Negative. The neutral responses to the Likert items were left as is. These are shown in Table 3 (Total number of responses=337).

Table 3: Summary of survey responses – Learning

Item	Positive (%)	Neutral (%)	Negative (%)
Scratch is useful for learning programming concepts.	70	18	12
It is useful for beginners to learn Scratch before moving on to C++.	69	15	16

The results show that a majority of students perceived Scratch to be useful for learning basic concepts as well as for transitioning to C++ (RQ2).

5.2 Engagement of Advanced Learners

5.2.1 Application of Programming Concepts: Scratch Projects

We were able to obtain 93 projects for analysis. In 35 out of 93, one or both members of the team had not taken the 'prior programming background' survey conducted in the first week (See section 4.1). Hence we could not classify these projects as belonging to novice or advanced learners. Of the remaining 58 projects, any team that had both novices was classified as a 'novice' project, while any team with at least one advanced learner was classified as an 'advanced' project. This resulted in 18 'novice' projects and 40 advanced projects. 13 out of the 40 advanced projects qualified as Hall-of-fame entries, so for analysis we split the advanced category into 27 'advanced' and 13 'Hall-of-fame'.

There was a variety of games in the projects. Some of the most frequent types of games were "shooting games", "maze based games", "batting games", and "car racing games". Six games were multi-player and thirty nine were single player games; forty were found to be single-level games while five had multiple level options. Each game was tested and found to be acceptable in terms of usability. First, we counted the number programming constructs used in novice, advanced and Hall-of-fame. These are shown in Table 4, along with corresponding numbers for 'Most-loved' projects from Scratch website [15], for reference. From Table 4, we note that advanced projects use on an average 67% constructs as the most-loved projects, while Hall-of-fame projects use 105% of constructs compared to Most-loved projects. If we combine the results of all advanced learners in our course, i.e., advanced and Hall-of-fame, we find that their constructs usage is 80% that of Most-loved projects.

Thus, our advanced learners are able to achieve results comparable to the Most-loved projects, even in a period of only three weeks. It should be noted that the reward for the Scratch project was just 5 marks, and even with this meagre reward students generated complex projects. This shows that they were indeed engaged.

Table 4: Average number of occurrences of programming constructs used in Scratch projects

Programming Constructs	'Novice' projects (N=18)	**'Advanced' projects (N=27)**	**Hall-of-fame projects (N=13)**	'Most loved games' (N=10)
Variables	5	**9**	**12**	9
Sprites	14	**30**	**35**	35
Stacks	38	**77**	**131**	101
Control	111	**230**	**417**	454
Motion	51	**83**	**147**	129
Operator	38	**92**	**147**	197
Sensing	32	**63**	**117**	75
Other blocks	38	**117**	**177**	403
Total	**327**	**701**	**1183**	**1403**

5.2.2 Student Perception of Engagement

The summary of responses corresponding to the Question 3 of the student perception survey questionnaire is shown in Table 5. (Total number of responses=337).

Table 5: Summary of survey responses - Engagement

Item	Positive (%)	Neutral (%)	Negative (%)
Enjoyed programming with Scratch.	65	15	20

As evident from Table 5, a majority of students perceived the experience of Scratch programming to be positive. This shows that Scratch has positively affected students' engagement.

6. DISCUSSION

We first examined if the Scratch intervention was effective in novices' acquisition of basic programming concepts and in their transition to C++ (RQ1). From the analysis of scores on exam questions, we find evidence of learning basic programming concepts. However, the performance of novice students was comparable to that of advanced learners only on Scratch questions of types: Predict output of a program, and debug a program. In typical questions of write a program, novices lag behind advanced learners. This is to be expected as only six weeks had elapsed in the course. The gap persists in scores on C++ questions. So from the perspective of achievement on traditional exams, we conclude that our Scratch intervention was only a qualified success.

When we analyze students' learning (RQ1) in terms of how well they apply programming concepts in the question-generation activity, we find that novices have a high 'catch-up' with advanced learners. In this non-traditional assessments, we see that i) novices indeed show strong evidence of application of programming concepts and ii) the performance of novices is often comparable to that of advanced students.

Student perception data from the survey strongly support that Scratch was useful for the learning of basic concepts of novices and that it helped them transition to C++ (RQ3).

The effectiveness of Scratch for advanced learners (RQ2) is clearly seen from the performance of advanced learners on Scratch projects, and is corroborated by survey data. The Scratch project creation activity pushed advanced students towards a high boundary in terms of the extent of their application of programming concepts. We see this as evidence of students'

engagement with the content and conclude that Scratch was an appropriate choice to address this instructional goal.

Open-ended responses in the survey data (Q4) support this point: students have said that "[I am] thrilled to be able to code complex games" and "[coding] games helped increase my interest, […], there was lot of room for experimentation." Responses to the first open-ended question in the survey data identified some major benefits. In addition to the cognitive aspect of the benefit of Scratch to learn programming concepts and skills, nearly a third of the responses addressed the affective benefit of using Scratch to begin programming. Several students commented that Scratch "helped improve confidence" and "removed fear of programming." Majority also commented that Scratch "helped in understanding C++", indicating its usefulness in transitioning to C++. Other benefits were the ease of programming threads and graphics, syntax-free visual environment and the fun element in using Scratch. On the contrary, responses to the second open ended question (Q5) brought out some disadvantages of using Scratch such as cumbersome features of the IDE and the limitation of the language capability of Scratch.

As mentioned in Section 1, our findings are subject to validity threats due to the field-study setting in which they were conducted. The biggest threat is the lack of control while interpreting exam scores. Since we could not compare the performance of our students with a group that did not get the Scratch intervention, it is possible that the learning we observe through exam performance could have been due to reasons other than the 2-week Scratch intervention. To offset this issue we used a combination of different data collection sources and different analysis approaches – qualitative and quantitative. We triangulated our findings from exam scores with several other sources. The variety of assessment instruments we used gave us different views into our research questions, and we were able to infer learning and engagement from several perspectives.

7. CONCLUSION

We conclude that an intervention of two weeks of Scratch lectures, along with three labs and a project is useful for learning basic programming concepts at CS1 level, addressing diversity by scaffolding for novices, engagement for advanced learners. However, this specific intervention of Scratch has limited usefulness in helping novices to catch up with advanced learners in typical programming exam questions.

Even though the original motivation for our study was to address the needs of the diverse academic population in an Indian university, our results are useful for: a) CS0/CS1 instructors looking for a solution to help students with no programming background from getting daunted by syntax and concepts, and, b) for engaging college students with prior programming exposure with building complex programs and exercising creative expression in programming.

8. REFERENCES

[1] Blackwell, A.F. 1996. Metacognitive Theories of Visual Programming: What do we think we are doing?. *Visual Languages, 1996. Proceedings*. IEEE Symposium on, Cambridge, Pages 240-246.

[2] Brennan, K., Resnick, M. 2012. New frameworks for studying and assessing the development of computational thinking. *In Proceedings of the 2012 annual meeting of the American Educational Research Association* (Vancouver, Canada).

[3] Carnegie Mellon University. Alicev2.0. www.alice.org.

[4] Csikszentmihalyi, M. 1997. Finding flow: The psychology of engagement with everyday life. Basic Books.

[5] Gray, S., Clair, S. C., James, R., Mead, J. Graduated exposure to programming concepts using fading worked examples. 2007. *ICER '07 Proceedings of the third international workshop on Computing education research.* Georgia, ACM, Pages 99-110.

[6] Home of Scrape. River Sound Media. happyanalyzing.com

[7] Jeliot3.Program Visualization tool. cs.joensuu.fi/jeliot/

[8] Lifelong Kindergarten, MIT Media Lab. Scratchweblogs. http://media.mit.edu/llk/scratch/.

[9] Malan, D. J., & Leitner, H. H. 2007. Scratch for budding computer scientists. ACM SIGCSE Bulletin, 39(1), 223-227.

[10] Margulieux, L. E., Guzdial, M., Catrambone, R. 2012. Subgoal-labeled instructional material improves performance and transfer in learning to develop mobile applications. In *Proceedings of the ninth annual international conference on International computing education research* (pp. 71-78). ACM.

[11] Meerbaum-Salant, O., Armoni, M., and Ben-Ari, M. M. 2010. Learning computer science concepts with scratch. In *Proceedings of the Sixth international workshop on Computing education research* (pp. 69-76). ACM.

[12] Reiser, Brian J. 2004. Scaffolding complex learning: The mechanisms of structuring and problematizing student work. *Journal of the Learning Sciences*: 13(3), 273-304.

[13] Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., and Kafai, Y. 2009. Scratch: programming for all. Communications of the ACM, 52(11), 60-67.

[14] Rizvi, M., Humphries, T., Major, D., Jones, M., and Lauzun, H. 2011. A CS0 course using scratch. Journal of Computing Sciences in Colleges, 26(3), 19-27.

[15] Scratch projects: scratch.mit.edu/tagged/toploved/game

[16] Sheard, J., Simon, S., Hamilton, M., Lonnberg, J. Analysis of Research into the Teaching and Learning of Programming. 2009. *ICER '09 Proceedings of the fifth international workshop on Computing education research workshop*. (California). ACM, Pages 93-104.

[17] Shernoff, D. J., Csikszentmihalyi, M., Shneider, B., and Shernoff, E. S. 2003. Student engagement in high school classrooms from the perspective of flow theory. School Psychology Quarterly, 18(2), 158-176.

[18] The JPie project. www.jpie.cse.wustl.edu/

[19] Van Merrienboer, J. J., & Sweller, J. 2005. Cognitive load theory and complex learning: Recent developments and future directions. Educational psychology review, 17(2), 147-177.

[20] Wolz, U., Leitner, H. H., Malan, D. J., and Maloney, J. 2009. Starting with Scratch in CS1. Proceedings - 40th *ACM technical symposium on CS education*, Tennessee, ACM, pp 2-3.