Delft University of Technology
Software Engineering Research Group
Technical Report Series

Teaching Software Engineering Principles to K-12 Students: A MOOC on Scratch

Felienne Hermans, Efthimia Aivaloglou

Report TUD-SERG-2017-003

TUD-SERG-2017-003

Published, produced and distributed by:

Software Engineering Research Group
Department of Software Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4
2628 CD Delft
The Netherlands

ISSN 1872-5392

Software Engineering Research Group Technical Reports:
http://www.se.ewi.tudelft.nl/techreports/

For more information about the Software Engineering Research Group:
http://www.se.ewi.tudelft.nl/

Note: -

Teaching Software Engineering Principles to K-12 Students: A MOOC on Scratch

Felienne Hermans and Efthimia Aivaloglou
Software Engineering Research Group
Delft University of Technology
Mekelweg 4, 2628 CD Delft, the Netherlands
f.f.j.hermans@tudelft.nl, e.aivaloglou@tudelft.nl

***Abstract*—In the last few years, many books, online puzzles, apps and games have been created to teach young children programming. However, most of these do not introduce children to broader concepts from software engineering, such as debugging and code quality issues like smells, duplication, refactoring and naming. To address this, we designed and ran an online introductory Scratch programming course in which we teach elementary programming concepts and software engineering concepts simultaneously. In total 2,220 children actively participated in our course in June and July 2016, most of which (73%) between the ages of 7 and 11. In this paper we describe our course design and analyze the resulting data. More specifically, we investigate whether 1) students find programming concepts more difficult than software engineering concepts, 2) there are age-related differences in their performance and 3) we can predict successful course completion. Our results show that there is no difference in students' scores between the programming concepts and the software engineering concepts, suggesting that it is indeed possible to teach these concepts to this age group. We also find that students over 12 years of age perform significantly better in questions related to *operators* and *procedures*. Finally, we identify the factors from the students' profile and their behaviour in the first week of the course that can be used to predict its successful completion.**

***Keywords*-Programming education, MOOC, Scratch, code smells, dropout prediction**

I. Introduction

Programming education, and the broader *computational thinking* [1], has been made part of the curriculum in many countries recently [2], [3]. However, most of the programming materials do not teach software engineering methods. We hypothesize that some fundamental software engineering methods can be taught to young children as part of an introductory programming course.

To explore this idea, we have developed an online introductory programming course in which not only programming concepts but also software engineering ideas are being instructed. In particular, we instruct students about code smells [4]: dead code, long method smell, duplication smell and bad naming smell, and train their debugging skills. In previous work we have found that the long method and the duplication smell are both harmful and common [5], [6].

Our course ran as a MOOC (massive open online course) on the edX platform from June 15th to September 1st 2016[1]. It included 6 weeks of course material, and remained available for another 4 weeks to give the opportunity to students that joined later to finish. The course lectures and all material were in Dutch. In total 3,179 students enrolled in the course, 2,220 participated in it, meaning they watched videos, submitted answers to quizzes or participated in the forum. Out of those, 181 successfully completed it, which is defined by edX as obtaining 60% of the possible points. This completion rate of 5.7% is in line with the average completion rate for MOOCs, which has been calculated to less than 7% [7], [8].

In this paper we present our course design, consisting of videos, mixed with formative quizzes, forum interactions and two summative exams, at the end of weeks 3 and 6. We further examine if children can understand, apart from programming concepts, concepts related to code quality. Moreover, because it has been found that specific programming concepts can be understood in certain ages [9], we want to examine whether we observe similar patterns in our course. At the same time, many papers in the domain of programming education have attempted to predict the factors related to students ceasing their participation in online courses. A number of factors that is found to contribute are student entry characteristics including students previous academic and professional experiences and performance, learning skills, and psychological attributes, supportive environments and encouragement, and students interactions within classrooms [10]. Using the data that became available through this course, and knowing that children exhibit different behavioral patterns than adults when following on-line courses [11], we wanted to investigate if the similar factors apply. Specifically, we are interested in examining which characteristics from the student profile and what type of participation and grading data from the first week of the course can be used for identifying the students that have potential for successfully finishing the course.

We analyze the data of all 2,220 students who started the course to determine 1) if students found the programming concepts more difficult than the software engineering concepts, 2) if there are age-related differences in their performance on the different concepts and 3) if we can predict course

[1]URL of course on self-paced mode: https://www.edx.org/course/scratch-programmeren-voor-kinderen-8-delftx-scratchx-0

completion based on information from the student profiles and their activities in the first week.

Our results show that there is no difference between the students' scores on the programming concepts and on the software engineering concepts. This gives credibility to the hypothesis that we can teach children about code quality and debugging. We also find that, for some concepts, there is a significant difference on the performance between 11-12 year old children with 13-14 year old ones. Finally, we are able to identify factors from the students' profile and their behaviour in the first week of the course that can be used to predict whether they will complete the (i.e. obtain 60% of the points).

The contributions of this paper are as follows:

- The course design of an online course for children teaching programming and software engineering methods
- An analysis of the difficulty of 7 programming and 5 software engineering based on quiz scores
- An machine learning model predicting factors for the successful completion of the course

II. Course Design

To explore the idea of teaching software engineering to children, we developed an online introductory programming course in Scratch. This course was marketed as an introductory programming course, we did not explicitly share the fact that software engineering methods would be taught as well.

During the course, we presented clean code and maintainability as things that programmers do and like. For example, when talking about dead code, we explained that programmers do not like a messy workspace, so that you need to clean your workspace.

A. Course overview

The course consists of six 'weeks', even though students are free to follow the course in their own pace. This is the default manner in which course content on edX is organized.

Each week consists of three components: videos, quizzes and forum interactions.

The videos are a combination of the screen as a whole and the instructor talking, are short in length and are presented with enthusiasm, as per the recommendations of Guo et al. based on 7 million viewing sessions [12].

In each week, the instructor creates a game in Scratch, which is shown in the videos. We expect students to follow along with the creation of the game, as presented in the videos. Often, the instructor creates part of the game and challenges students to finish the remainder, for example, demonstrating how to control a sprite to move left, and encouraging the students to complete similar code for right, up and down.

In between these programming assignments, students get quizzes in which their knowledge is tested. These quizzes are sometimes related to the concrete programming assignment that has given, like what code is needed to make the sprite move right. These quizzes were mixed with quizzes testing students' general knowledge of programming concepts.

The quizzes are designed as *formative* assignments, giving students feedback on their learning progress. Students are allowed two attempts for each quiz and get encouraging feedback on wrong submissions, like 'that was close, but maybe you need to make one little change'.

In addition to the videos and quizzes, we encouraged students to explore the forum. The forum can be used to ask questions if they are stuck, but also to chat informally about their progress. By asking students specifically to participate in the forum, for example by asking children to share their specific approach with us, we hoped to motivate them to also ask us questions in case they needed help. During the run of the course, we monitored the forum almost every day.

B. Course Contents

Our course differs from other introductory programming courses, by emphasizing software engineering concepts related to source code quality. As such, the contents of the course are both aimed at programming concepts and on the additional software engineering methods that students are taught about. Table I presents an overview of the course contents.

1) Week 1: In the first week, students create a simple game in which the player navigates a sprite through a maze. In this week, we explain the Scratch basics of sprites and their blocks, and we introduce students to control flow: in the form of loops and if-then-else. We also show them how to store data in a variable. In a case where the instructor creates dead code, by moving a block from one script to the other, we explain that programmers do not like a messy workspace, and that needs to be cleaned up.

2) Week 2: Students create a game where a fish eats smaller fishes to grow bigger himself. We explain to students that programmers prefer small scripts over longer ones. We even share the fact our research has shown that shorter scripts are easier to read. In this week, procedures also make their first appearance. When we explain procedures, we explain that we want to minimize repetition. We furthermore explain the functionality of the `random number` block.

3) Week 3: We let students create a quiz on their favourite topic. The main topic of this week is the introduction of lists as variables, with the corresponding operations for adding items to a list, removing them and accessing specific items. We also repeat random numbers, conditionals and loops, and introduce conditional loops. This week also marks the start of the topic of debugging, where we deliberately make mistakes in programming and have students decide if we are correct or not, and have them fix the instructor's "mistakes".

Week 3 introduces fewer new concepts, as it also contains the intermediate exam discussed in Section II-C.

4) Week 4: Students program a "crossy road" game, in which a dog needs to avoid traffic to reach the other side of a road on which food items are placed. Again we repeat loops and conditionals already taught, we also have students debug programs again, and let students practise with creating procedures to minimize duplication. In addition to that, we explore the XY plane more, learn about using the `x-pos` and

TABLE I
OVERVIEW OF THE CONCEPTS TAUGHT IN EACH SECTION OF THE COURSE

Topic	Summary of Course Material
	Week 1
Conditionals	The if-then-else block is introduced, as well as the possibility for sprite to touch each other
Events	We introduce the when-key-pressed and blocks to move, turn, and control the speak bubbles of sprites
Variables	We introduce the concept of variables
Dead code	We teach children to remove unused blocks, as programmers do not like clutter
Loops	We explain the do-forever loop
	Week 2
Conditionals	We repeat the if-then else block
Events	Blocks to show, hide and shrink sprites
Variables	We repeat variables and introduce random numbers
Coordination	We introduce broadcast-receive blocks
Duplication	We show repeated scripts and explain that we want to minimize this
Long method	We show a long script and explain that this is less readable than two separate scripts
Loops	We repeat the use of the do-forever block
Naming	We encourage students to select a good name when naming a signal
Procedures	We show children how to create their own block and explain that this can be used to minimize duplication
	Week 3
Conditionals	We repeat the if-then-else block
Events	We explain how sprites can ask and use the answer
Variables	We use lists and related concepts: add, empty and get item
Debugging	From week 3 on, we deliberately make mistakes in programming and challenge children to find them
Loops	The do-until block is taught
	Week 4
Conditionals	We repeat the if-then else block
Events	We introduce measuring distance to other sprites
Variables	We introduce using x-pos and y-pos as variables
Debugging	From week 3 on, we deliberately make mistakes in programming and challenge children to find them.
Duplication	Students have to decide how to create blocks to minimize duplication
Loops	We repeat all loops previously taught
Operators	We explain >and <
Parallelization	We explain the {wait} block and the concept of blocking operations.
Procedures	Students have to decide how to create custom blocks to minimize duplication
	Week 5
Conditionals	We repeat previously taught conditionals
Events	Introducing the use of sounds and the webcam
Variables	More practise with variables including random variables
Debugging	From week 3 on, we deliberately make mistakes in programming and challenge children to find them.
Loops	We repeat all loops previously taught
Operators	We explain < and > and their use.
	Week 6
Conditionals	The operator & is introduced to combine conditionals
Events	Explaining how to stop scripts
Variables	Exercises use variables
Debugging	From week 3 on, we deliberately make mistakes in programming and challenge children to find them.
Duplication	Students have to decide how to create blocks to minimize duplication
Loops	The do-repeat block is introduced
Naming	A new sprite created by Scratch is called 'sprite1'. We explain why that is not a preferred name
Operators	Student practise more with >and <
Procedures	More practise with creating blocks

TABLE II
EVALUATED PROGRAMMING AND SOFTWARE ENGINEERING CONCEPTS

Concept	Example question
Conditionals	We are going to determine if the answer is correct. What programming block do we need?
Coordination	The fish needs to grow when he catches a purple fish. How do you program that?
Variables	We created the variable 'speed' and set it to 0. Now it needs to decrease. What block to use?
Loops	When does the wheel need to stop spinning?
Operators	We will check two things: whether Giga touches the wall and ...? What do we need to add?
Parallelism	The bug disappears at the beginning of the game, but moves when Giga moves. Which script achieves this?
Procedures	Can you create your own countdown block? Which blocks will go in it?
Dead code	What will happen when we execute this code?
Debugging	We have created this program. Does it work?
Duplication	This code is correct, but, there is one block we could remove. Which one?
Long method	That's a lot of blocks. Can you make the code more readable?
Naming	Describe the steps needed to cross the streets in Dutch. What parts of the code do they relate to?

`y-pos` as properties of sprites and for measuring distance between sprites. Finally, we introduce the `wait until` block and explain blocking operations.

5) Week 5: Students build a "flappy bird" game that children can control with the webcam. The main goal of this week is to amaze students with the possibilities that programming brings, in addition to repeating previously taught concepts. We further practice using $<$ and $>$ operations.

6) Week 6: In this week, students make the capstone game of the course, a Mario-like platformer. In this week all skills of the previous weeks are needed, the only new concept introduced is the `do repeat` block, and we again stress the importance of good names, this time for sprites and costumes.

C. Evaluation and Feedback

To gain feedback on the progress of students towards learning about programming concepts we used two types of tests: the quizzes that were included in every week of the course and were designed to assist the students in understanding the taught concepts, and the interim and final tests at the end of the third and the sixth week respectively. The course included a total of 64 quizzes with a total of 117 multiple choice questions, while the 2 tests included 55 questions.

The questions of the quizzes and the tests were designed to correspond to specific programming concepts. We used the 7 programming concepts similar to the ones commonly utilized in the literature ([13], [14], [9], [15]). They are outlined, along with example questions, in Table II. The questions of the 2 tests were designed to cover the six categories of the Bloom taxonomy, as applied to computational thinking in [16] and [17]. There were cases of questions testing more than one programming concept at the same time. The students had the opportunity to try each question twice and they got directions after the first try.

III. Data and Research Design

With the data of the course, we aim to answer the following resesarch questions:

- RQ1 Do students perform better on the questions related to the 7 programming concepts than on the 5 related to code quality?
- RQ2 Are there age-related differences in the students' performance on all 12 concepts?
- RQ3 Can we predict course completion based on information from the student profiles and their activities in the first week?

This section describes the dataset on which we perform our analyses, as well as the approach we use to answer the three research questions.

A. Student backgrounds

In total 3,179 students enrolled in the course, while 2,220 actively participated in it by watching at least one video or submitting at least one answer to a quiz or participating in the course discussion forum. 181 students successfully completed it, defined by edX as obtaining 60% of the possible points.

In order to understand our course population better, we are interested in the age and the gender of the students, the programming experience that they already have, and whether they are receiving off-line support during the course. Regarding age and gender, this could be obtained from the user profile on the edx platform. This information however might not be representative of the course students, because they could be using their parents's user profiles to participate to the course. This is somewhat likely, since edX is a platform aimed at adult professional learners rather than at children.

For this reason, we ask students about their age and gender in a questionnaire, as the first task on Week 1 of the course. In the same questionnaire we ask them about their previous programming experience and whether they are following the course alone or have some off-line support.

Of the 2,220 active students in the course, 1,243 filled in this student profile questionnaire.

The *age distribution* of the students is plotted in Figure 1. Using their ages as they reported them in the questionnaire, most students (73.71% of the 1,202 that filled in their age at the questionnaire) are 7 to 11 years old, with the most popular ages being 8 (for 19.97% of the 1,202 students) and 9 (for 17.30% of them). Using the edX user account information, the most popular ages are 9, 10 and 11, together accounting for 34.11% of the 2,553 users that registered for the course and have filled in their year of birth to their edX profile. At the same time, 40.03% of those edX users are over 20, which could indicate students using their parents accounts, but also contains some parents and teachers sampling the course to decide whether it is fun and educational. Though email and Twitter we received a few messages from teachers along those lines.

Out of the 1,093 students that reported their *gender* at the questionnaire, 346 (31.66%) are female. The gender distribution is similar in the edX accounts, where 891 (35.13%) of the 2,536 users that have provided their gender are female.

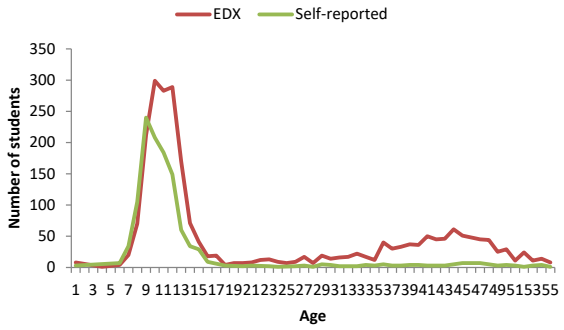

Fig. 1. Age distribution of the students participating in the course

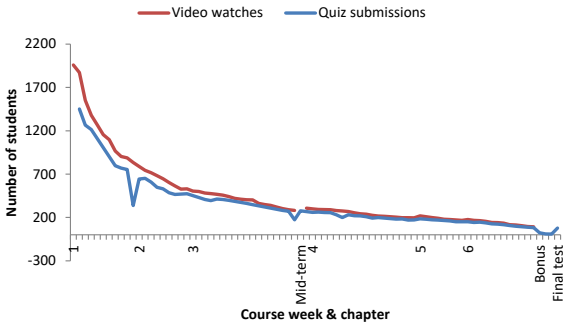

Fig. 2. Student engagement in course videos and quizzes

Asked about their *programming experience*, the majority of the students (58.15% of the 1,147 that answered this question at the questionnaire) answered that they have none, while 19.09% reported having used Scratch in the past and even 4.79% had programmed with Lego Mindstorms.

Regarding *off-line support*, the majority of the students (60.21% of the 1,141 that answered this question at the questionnaire) reported following the course at home together with their father, mother or some other adult. In-school teacher support was provided to 5.43% of the students, while a few students (3.33%) followed the course together with a friend or sibling, and the rest followed it alone.

B. Student Behaviour

The number of students engaged in the course throughout its time line is plotted in Figure 2.

The first *video* in the first week of the course was watched by 1,958 students (88.2% of the 2,220 active participants), which dropped to 788 students (35.49% of the active ones) in the second week. The sixth and final week of the course started with 176 students (7.93% of the active ones) and finished with 93. Examining the video interactions, the most common behavior is pausing the video streams, in 18.71% of the 102,133 video watches. Video pausing behavior is notable: all 19,110 paused video watches included more than

	total	mean	min	Q1	median	Q3	max
Duration (seconds)		52.60	6	16	34	70	1,794
Forward seek (times)	6,333	0.16	0	0	0	0	37
Backward seek (times)	5,886	0.11	0	0	0	0	43
Speed up (times)	258	0.00	0	0	0	0	1
Pause (times)	19,110	6.87	0	0	0	0	614

TABLE III
SUMMARY STATISTICS OF VIDEO INTERACTIONS CALCULATED OVER 102,133 VIDEO WATCHES

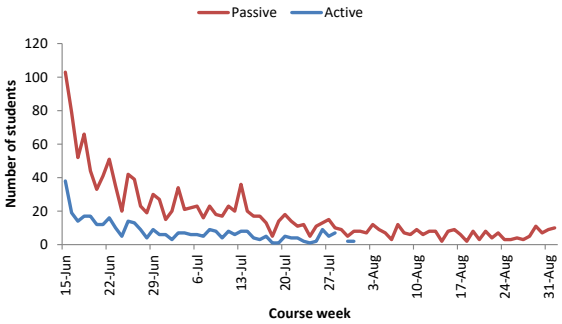

Fig. 3. Use of the course discussion forum

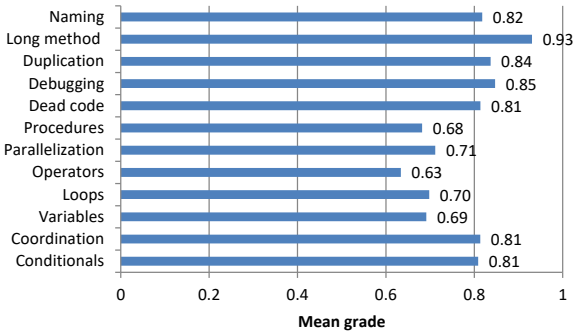

Fig. 4. Mean grade of all student answers (first attempts) to all questions corresponding to different programming and software engineering concepts

5 pauses, while 10,572 of them included more than 20. Because most of the videos in the course walk the students through the implementation of Scratch programs, we assume that this video pausing behavior indicates that students would simultaneously program in the Scratch web interface while watching the instructions. Table III summarizes all types of video interactions during the course.

The first *quiz* of the course was the student profile questionnaire discussed in the previous section. The first quiz on the course material was answered by 1,452 students (65.4% of the active ones). Submissions for the mid-term *exam* at the end of week three were received by 276 students out of the 281 that were still in the course and had watched the final video of that week.

The *discussion forum* of the course was used by 706 students (31.8% of the active ones). 200 of them actively participated on it, submitting a total of 691 posts and replies in 195 different threads, while the rest were passive forum users, visiting it and possibly reading posts and searching —a total of 76 forum searches were recorded during the course. Figure 3 summarizes the number of active and passive forum users during the course.

C. Data analysis

To answer *RQ1*, we obtained feedback on the progress of students towards learning about programming concepts through the quizzes and tests discussed in Section II-C. While the students had the opportunity to try each question twice, in the analysis we use only the assessment for their first attempt, since students got directions after the first try. In each question, the students are graded with 1 if it is correct or 0 otherwise. All questions were assigned the same weight, so the mean student grade is the same as the mean correct answers.

We investigate the difference between the grades that the students received in programming and in software engineering questions. We created two sets, one with their mean test results in programming questions and one with the ones in software engineering questions. We use the Wilcoxon rank sum test to compare the two sets and Cliff's delta to calculate the effect size.

Related to *RQ2*, Seiter *et al.* observe that certain concepts are too advanced for most students within primary school grades [9]. Since our data seemed to follow a similar pattern, we compared groups older and younger than 12 respectively.

We therefore created for each tested programming concept two sets, one with the test results of the 11 and the 12 year old students and one with the results of the 13 and 14 year olds. We used the age of the students as either the one they declared in the student profile questionnaire or, if they did not fill it in, we calculated it from their edX account information. We then used the Wilcoxon rank sum test to compare the two sets and Cliff's delta to calculate the effect size.

For *RQ3*, we are interested in examining which characteristics from the student profile and what type of participation and grading data from the first week of the course can be used for identifying the students that have potential for successfully finishing the course. To understand the factors that lead to course competition or dropout, we use machine learning. We generate the dataset of the features that are described in Table IV for all students that have watched at least one video or have submitted at least one answer to a question of the first week of the course. We model retention as a binary classification problem, where given those features, we try to predict whether

the student will score over 60% on edX's cumulative scoring mechanism.

At a high level, the process to retrieve the dominant factors that affect retention for the classification task consists of two steps. First, we run the dataset through 2 classification algorithms, namely Random Forests and Binary Logistic Regression. To evaluate the classification performance, we use Area Under the Receiver Operating Characteristic Curve (AUC) and F1 metrics. Both metrics are typical for binary classification tasks, especially when comparing model performance. To select the appropriate classification algorithm, we run a 10-fold random selection cross-validation and aggregate the mean values for each classification metric. At each iteration, the algorithm randomly samples half of the available data points, trains a classifier with 80% percent of the input and uses it to predict the remaining 20%. The 10-fold run results also allowed us to evaluate the metric stability across runs. We did not perform any hyperparameter tuning at this stage. The results showed that both Logistic Regression and Random Forests achieve an AUC score of 0.84 and a comparable F1 score of 0.49 and 0.48 respectively. As Binary Logistic Regression offers higher explainability of the effect of factors on the outcome and comparable performance scores, we dropped Random Forests from further examination.

Having selected Binary Logistic Regression, we further tune our model. To do so, and for each Task, we build 5 variants for our classification models:

1) *V*1: is the default model.
2) *V*2: like V1, with highly correlated (ρ threshold: 0.8) parameters eliminated.
3) *V*3: like V2, adding a check for multicollinearity using the the Variance Inflation Factor (VIF).
4) *V*4: like V1, but to stabilize variance, we apply a log transformation to numerical values.
5) *V*5: like V1, but we apply a log transformation to numerical values after removing highly correlated values

We cross-compare the variants using the Akaike information criterion (AIC) score, which provides a relative estimate of the information lost when a given model is used to represent the process that generates the data. We supplement the comparison with each model's deviance. Moreover, to evaluate model fit, we compare all variants in terms of AUC. The results can be seen in Table V.

IV. Results

A. RQ1: Do students perform better on the questions related to the 7 programming concepts than on the 5 related to code quality?

Figure 4 summarizes the results obtained from a total of 52,264 student answers to the quiz and test questions of the different categories. Students generally performed better at the quiz questions, which is expected because they were designed to be formative rather than summative. The lowest percentage of correct answers was given for the Operators, Procedures and Variables categories.

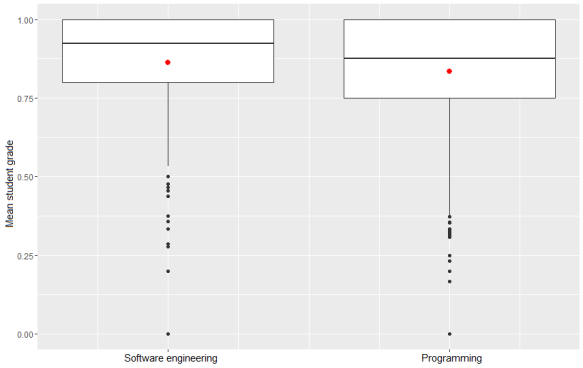

Fig. 5. Mean grade per student on questions related to software engineering concepts and to programming concepts

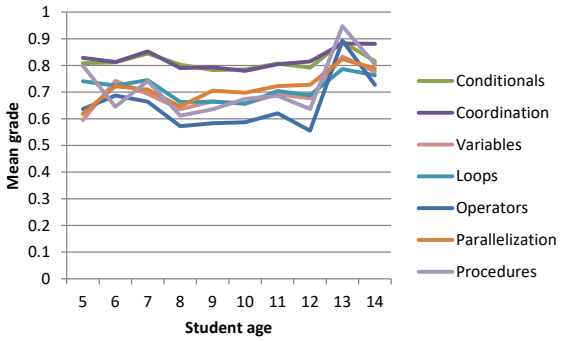

Fig. 6. Answer correctness per student age and tested programming concept

Comparing the performance of students on questions related to software engineering concepts with their performance on programming concepts, we found that the difference is not significant. Figure 5 shows the variance of the mean grade per student in the two types of questions. The calculated p-value is very small (8.301e-08), but the effect size of 0.141 is too small to characterize the difference between the two groups as significant.

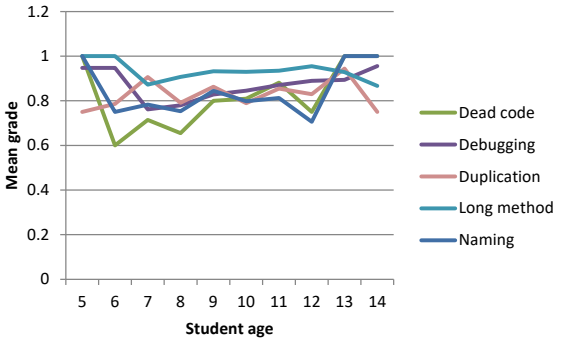

Fig. 7. Answer correctness per student age and tested software engineering concept

Feature	Description	Quant_5	Mean	Median	Quant_95	Histogram
gender	The gender declared in the student profile questionnaire	-	-	-	-	
age	The age declared in the student profile questionnaire or the edX profile age	0.00	14.71	10.00	50.00	
alone	Is the student following the course alone? (from the student profile questionnaire)	-	-	-	-	
experience	Does the student have prior programming experience? (from the student profile questionnaire)	-	-	-	-	
joined_calendar_week	Which week was the course in when the student first became active	1.00	2.88	2.00	7.00	
forum_searches	Number of searches in the discussion forum in the 1st course week	0.00	0.01	0.00	0.00	
forum_times_accessed	Times the discussion forum was accessed in the 1st week	0.00	0.17	0.00	1.00	
forum_max_duration	Duration of the longest session in the forum in the 1st week (seconds)	0.00	19.95	0.00	59.00	
distinct_videos_watched	Different 1st week videos watched	0.00	3.14	0.00	15.00	
prc_videos_warched	Percentage of 1st week videos watched	0.00	0.20	0.00	1.00	
skipped_video	Did the student skip any of the 1st week videos?	-	-	-	-	
total_watches	Total number of video watches in the 1st week	0.00	7.75	0.00	43.00	
days_engaged_videos	Distinct calendar days engaged in 1st week video watching	0.00	0.60	0.00	3.00	
total_pauses	Number of video pauses in the 1st week	0.00	53.19	0.00	365.00	
total_forward_seek	Number of video forwards in the 1st week	0.00	1.21	0.00	7.00	
total_backward_seek	Number of video backwards in the 1st week	0.00	0.85	0.00	6.00	
mean_pauses	Mean pauses per video in the 1st week	0.00	1.78	0.00	11.00	
mean_forward_seek	Mean forwards per video in the 1st week	0.00	0.05	0.00	0.30	
mean_backward_seek	Mean backwards per video in the 1st week	0.00	0.03	0.00	0.21	
total_duration	Total time spent watching 1st week videos	0.00	407.48	0.00	2145.05	
mean_duration	Mean time per video in the 1st week	0.00	19.25	0.00	91.47	
questionnaires_skipped	Number of 1st week questionnaires skipped	0.00	9.14	8.00	17.00	
questonnaires_tried	Number of 1st week questionnaires with at least one submitted answer	0.00	1.70	0.00	8.00	
questions_skipped	Number of 1st week questions skipped	0.00	20.01	16.00	47.00	
questions_tried	Number of 1st week questions with submitted answer	0.00	3.83	0.00	18.00	
mean_tries_question	Mean submissions per 1st week question	0.00	0.38	0.00	1.77	
mean_grade	Mean grade from the 1st week questions	0.00	0.25	0.00	1.00	
has_failed_answer	Did the student fail in any of the 1st week questions?	-	-	-	-	

TABLE IV
SELECTED FEATURES AND DESCRIPTIVE STATISTICS. HISTOGRAMS GIVE AN OVERVIEW OF THE DATA. THEY ARE IN LOG SCALE.

Model variant	**V1**	**V2**	**V3**	**V4**	**V5**
AIC	593	605	608	**593**	693
Deviance	529	559	566	**525**	663
AUC	0.81	0.781	0.77	**0.84**	0.71

TABLE V
MODEL SELECTION RESULTS

B. RQ2: Are there age-related differences in performance on all 12 concepts?

An overview of the mean grades per student age and tested concept is presented in Figures 6 and 7, while the comparison results between the sets of 11-12 and 13-14 year old student results are listed in Table VI.

Our findings indicate that the two sets differ significantly ($p < 0.05$ and effect size > 0.4) for the case of Operators and, especially, Procedures, with effect sizes of 0.468 and 0.615 respectively. The p-value is also small for the case of Conditionals, Coordination and Variables, but the effect size is too small. For the software engineering concepts that the students were tested on, the difference between the two age

	p-value	**Effect size**
Conditionals	0.003	0.236
Coordination	0.008	0.187
Variables	0.021	0.220
Loops	0.282	0.081
Operators	**0.031**	**0.468**
Parallelization	0.586	0.073
Procedures	**0.008**	**0.615**
Dead code	0.337	0.205
Debugging	0.149	0.168
Duplication	0.646	0.044
Naming	0.067	0.393

TABLE VI
P-VALUES AND EFFECT SIZES (CLIFF'S DELTA) OF THE WILCOXON RANK SUM TEST BETWEEN THE GRADES OF 11-12 YEAR OLD STUDENTS AND THE ONES OF 13-14 YEAR OLD STUDENTS.

	Model 1
(Intercept)	−13.58 (27.58)
genderBoy	−0.53 (1.24)
genderGirl	−0.63 (1.25)
log(age + 1)	−0.18 (0.30)
aloneAt school with help from a teacher	−0.80 (0.85)
aloneTogether with a friend or sibling	−1.18 (0.82)
aloneAt home with help from parent	−0.57 (0.27)*
experienceYes, with Scratch	0.88 (0.49)
experienceYes, with other	0.05 (0.53)
experienceNo	−0.17 (0.49)
log(joined_calendar_week)	−0.44 (0.18)*
log(forum_searches + 1)	0.25 (1.23)
log(forum_times_accessed + 1)	−0.39 (0.42)
log(forum_max_duration + 1)	0.13 (0.09)
log(distinct_videos_watched + 1)	3.20 (9.60)
log(prc_videos_warched + 0.01)	−2.43 (8.11)
skipped_video1	0.47 (0.66)
log(total_watches + 1)	0.21 (2.55)
log(days_engaged_videos + 1)	0.52 (0.38)
log(total_pauses + 1)	−0.23 (0.25)
log(total_forward_seek + 1)	0.05 (0.45)
log(total_backward_seek + 1)	0.28 (0.50)
log(mean_pauses + 1)	0.63 (0.47)
log(mean_forward_seek + 0.01)	−0.03 (0.30)
log(mean_backward_seek + 0.01)	−0.10 (0.32)
log(total_duration + 1)	−0.31 (2.31)
log(mean_duration + 0.01)	0.27 (2.37)
log(questionnaires_skipped + 1)	0.01 (0.96)
log(questonnaires_tried + 1)	7.28 (3.22)*
log(questions_skipped + 1)	0.04 (0.59)
log(questions_tried + 1)	−4.06 (3.27)
log(mean_tries_question + 0.01)	−0.19 (0.55)
log(mean_grade + 0.01)	21.82 (9.84)*
has_failed_answer1	1.69 (0.74)*
AIC	593.02
BIC	762.59
Log Likelihood	−262.51
Deviance	525.02
Num. obs.	1083

$^{***}p < 0.001$, $^{**}p < 0.01$, $^{*}p < 0.05$

TABLE VII
LOGISTIC MODEL FOR PREDICTING WHETHER A STUDENT WILL REACH EDX SCORE $> 60\%$ USING PROFILE & 1ST WEEK INTERACTION DATA

groups was not significant.

C. RQ3: Can we predict course completion based on information from the student profiles and their activities in the first week?

In the model of Table VII we see that following the course "at home with help from parent" and being late in joining the course reduces the chance of successfully completing it. At the same time, the factors that have a statistically significant positive influence are the number of questionnaires that have been submitted, the mean grade in the quizzes and having a failed answer.

Table VIII shows the deviance explained by each factor. Apart from the aforementioned factors, it shows that the experience, the number of times that the discussion forum was accessed, the distinct videos that were watched and the number of skipped questionnaires also explain a small, yet significant percentage of the total deviance.

The results also reveal that the effect of the gender is statistically insignificant, meaning that there are no differences between boys and girls in their chances of successfully completing the course. The same applies to their video interactions (pauses, forward and backward seek, total time spent on course videos). Moreover, even though the age of students affects their performance in quizzes and tests, as discussed in Section IV-B, the age feature does not have a significant effect on their chances of completing of the course.

V. DISCUSSION

A. Threats to validity

The first threat to the validity of our findings pertains to the self-selection of the course by the participants, because we cannot assume that the children in our course are a random sample of the Dutch population. Instead they, or their parents, decided to participate in an online programming course. Therefore, it is likely that these children are motivated to learn, and as such we will observe patterns that might not hold for all students. This threat will be addressed in future work, as currently over 10 classes are using our materials in a classroom setting for ages 9 and up. Furthermore, not all students filled out the age and gender questions in week 1. Out of the 2,220 participants 1,452 filled out the survey. In our data analysis as described in III-C, we used the age on the edX platform in case this data was missing, which might belong to the parent or teacher of the student and hence not be representative.

B. Naming

Previous work has shown that children rarely change the default names of sprites [18], which is one of the code smells that this course aims to educate about. While there is too little data to measure an effect, a first manual sample of our students' program does show that they changed the names of a new sprite created in week 6. However, this was part of the course materials, and it remains to be seen whether the students would also exhibit this behaviour while programming "in the wild". An analysis of the programs that students create after this course is a direction for future work.

C. Difficult programming concepts

Other papers [9] have observed that some programming concepts, especially variables and conditionals, are harder for certain age groups. While we do not measure significant differences for conditionals and variables, we do see an increase in scores on those two categories in our data as well. Our finding that younger children (less than 12 years old) perform worse in questions related to Procedures is an indication that this is an advanced concept and may be the reason why it is rarely applied in Scratch projects —only 8% of 250 thousand projects in the Scratch repository are found to utilize custom blocks [6]. Finally, it is notable that the threshold of 12 years of age that we found for increased performance at Operators and Procedures is also the age after which the children in the Netherlands move from elementary school to high school— 88.91% of the course participants declares the Netherlands as their country of origin in their edX profiles. An interesting direction for future work would be to explore whether their increased performance on those programming concepts is due to the more advanced mathematical concepts that they are taught within their high school curriculum.

	Df	Deviance	Resid. Df	Resid. Dev	Pr (>Chi)
NULL			1082	741.60	
gender	2	2.12	1080	739.48	0.3464
log(age + 1)	1	0.04	1079	739.44	0.8401
alone	3	6.61	1076	732.83	0.0853
experience	3	32.70	1073	700.13	0.0000
log(joined_calendar_week)	1	11.63	1072	688.50	0.0007
log(forum_searches + 1)	1	0.00	1071	688.50	0.9818
log(forum_times_accessed + 1)	1	15.02	1070	673.48	0.0001
log(forum_max_duration + 1)	1	3.26	1069	670.23	0.0710
log(distinct_videos_watched + 1)	1	61.01	1068	609.21	0.0000
log(prc_videos_warched + 0.01)	1	26.10	1067	583.11	0.0000
skipped_video	1	0.53	1066	582.59	0.4684
log(total_watches + 1)	1	0.05	1065	582.54	0.8296
log(days_engaged_videos + 1)	1	1.82	1064	580.72	0.1771
log(total_pauses + 1)	1	1.91	1063	578.81	0.1665
log(total_forward_seek + 1)	1	0.40	1062	578.40	0.5253
log(total_backward_seek + 1)	1	0.15	1061	578.25	0.6943
log(mean_pauses + 1)	1	1.20	1060	577.05	0.2738
log(mean_forward_seek + 0.01)	1	0.01	1059	577.04	0.9125
log(mean_backward_seek + 0.01)	1	0.21	1058	576.83	0.6494
log(total_duration + 1)	1	0.01	1057	576.82	0.9194
log(mean_duration + 0.01)	1	1.65	1056	575.17	0.1992
log(questionnaires_skipped + 1)	1	27.89	1055	547.28	0.0000
log(questonnaires_tried + 1)	1	10.38	1054	536.89	0.0013
log(questions_skipped + 1)	1	2.26	1053	534.63	0.1327
log(questions_tried + 1)	1	1.38	1052	533.26	0.2407
log(mean_tries_question + 0.01)	1	0.21	1051	533.05	0.6455
log(mean_grade + 0.01)	1	1.50	1050	531.55	0.2212
has_failed_answer	1	6.53	1049	525.02	0.0106

TABLE VIII
DEVIANCE EXPLAINED BY FACTORS AFFECTING RETENTION

VI. RELATED WORK

A number of studies have been carried out on teaching programming concepts to novice programmers with block-based languages in general, and Scratch in particular. Scratch was taught in middle school classes containing a total of 46 students in the study presented in [14]. Evaluating the internalization of programming concepts, it was found that students had problems with concepts related to initialization, variables and concurrency. In [18], Wilson et al. present an 8-week Scratch course given to 4 primary school classes with a total of 60 students aged 8 to 11, and evaluate it by analyzing the 29 projects that the students created. Maloney et al. [13] taught Scratch as an extracurricular activity, in an after-school clubhouse. By analyzing the 536 students' projects for blocks that relate to programming concepts, they found that within the least utilized ones are boolean operators and variables.

Apart from projects created during courses, other works analyze the public repository of Scratch programs for indications of learning of programming concepts. Yang et al. examined the learning patterns of programmers in terms of block use over their first 50 projects [19]. In [15], the use of programming concepts was examined in relation to the level of participation, the gender, and the account age of 5 thousand Scratch programmers. Moreno and Robles analyzed 100 Scratch projects to detect bad programming habits related to Naming and Duplication [20]. In prior work, we have analyzed 250 thousand Scratch projects in terms of complexity, used programming concepts and smells [6]. Seiter and Foreman [9] proposed a model for assessing computational thinking in primary school students and applied it on 150 Scratch projects, finding that design patterns requiring understanding of parallelization, conditionals and, especially, variables were under-represented by all grades apart from 5 an 6.

The topic of student retention and dropout in MOOCs has received significant attention during the last years. Lee and Choi have reviewed 35 empirical studies on students dropout in post-secondary on line courses and categorized the 44 common dropout factors into student factors, course/program factors, and environmental factors [10]. More recently, Kloft et al. [21] used machine learning to predict MOOC dropout over weeks using clickstream data from the page view log and the lecture video log. [22] presents a dropout predictor that uses student activity features, including video views ans exams taken, to predict which students have a high risk of dropout. Similar to our analysis, data from the first week of a university on-line course was used in [23] to predict the students performance at the end of the course and it was found that assignment performance in Week 1 is a strong predictor.

While a large volume of research has been carried out on postsecondary education MOOCs, limited research has been published on online courses aimed for K-12 schoolage children. A MOOC on Scratch programming has been taught in Uruguay; [24] presents the course description and initial results. In [25], a university-level CS1 course is offered as a MOOC to school children over 15 years old and they find that upper secondary school participants in the MOOC perform as well as the older participants. In [11], Yin et al. report that interviews of children-parent couplets having completed at least one MOOC revealed that children perceive MOOC video lectures differently than adults, that family interactions can affect the course experience, and that children may play with the MOOC material more than adults.

VII. Conclusion

The goal of this paper is to share our experience with our course on software engineering for children and the corresponding data analysis. More specifically, we study 1) whether students found the 7 programming concepts more difficult than the 5 software engineering concepts, 2) if there are age-related differences in their performance on the different concepts and 3) if we can predict course completion based on information from the student profiles and their activities in the first week.

Our results show that there is no difference in students' scores on the programming concepts and the software engineering concepts. We also find that for operators and procedures there is a significant difference between the performance of students below 12 years old with the performance of the older ones. Finally, we find that being late in joining the course and following it at home with a parent or other adult reduces the chance of successfully completing it, submitting answers to all questionnaires and a high mean grade in the first week are positive factors, while the age and gender have no significant effect in successful course completion.

We hypothesize that a course like ours could help students think about maintainability of source code from their first experience. While our results give some credibility to this hypothesis, quizzes in a course are not sufficient proof. Therefore, one of the directions for future work that we envision is to follow students who took our course in their programming career, to measure if there is indeed a difference. A concrete plan we have in this direction is to sample Scratch programs of our participants in a few months, and compare the quality of their programs in terms of the smells we taught to the Scratch programs of the general population.

Acknowledgments

We would like to thank Georgios Gousios for his help with data analysis, as well as Claudia Hauff and Yue Zhao for their help with obtaining the course data. Also, we would like to thank everyone who helped review the course material, especially the students of Instituut Het Centrum in Rotterdam: Darko Donker and Nickolay Frissen, and our teaching assistants who tirelessly answered questions on the forum: Jesse Donkervliet and Stefan Hugtenberg. Finally we thank our cameraman Jan Douma who helped us record the MOOC videos.

References

[1] J. M. Wing, "Computational thinking," *Commun. ACM*, vol. 49, no. 3, pp. 33–35, Mar. 2006.

[2] H. Hong, J. Wang, and S. H. Moghadam, "K-12 computer science education across the U.S." in *9th International Conference on Informatics in Schools: Situation, Evolution, and Perspectives*. Springer International Publishing, 2016, pp. 142–154.

[3] E. Barendsen, N. Grgurina, and J. Tolboom, "A new informatics curriculum for secondary education in The Netherlands," in *9th International Conference on Informatics in Schools: Situation, Evolution, and Perspectives*. Springer International Publishing, 2016, pp. 105–117.

[4] M. Fowler, *Refactoring: improving the design of existing code*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[5] F. Hermans and E. Aivaloglou, "Do code smells hamper novice programming? a controlled experiment on scratch programs," in *2016 IEEE 24th International Conference on Program Comprehension*, 2016, pp. 1–10.

[6] E. Aivaloglou and F. Hermans, "How kids code and how we know: An exploratory study on the scratch repository," in *Proceedings of the 2016 ACM Conference on International Computing Education Research*, ser. ICER '16. ACM, 2016, pp. 53–61.

[7] C. Parr. (2013) Not staying the course. [Online]. Available: https://www.insidehighered.com/news/2013/05/10/new-study-low-mooc-completion-rates

[8] K. Jordan. Mooc completion rates: The data. [Online]. Available: http://www.katyjordan.com/MOOCproject.html

[9] L. Seiter and B. Foreman, "Modeling the learning progressions of computational thinking of primary grade students," in *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research*. ACM, 2013, pp. 59–66.

[10] Y. Lee and J. Choi, "A review of online course dropout research: implications for practice and future research," *Educational Technology Research and Development*, vol. 59, no. 5, pp. 593–618, 2011.

[11] Y. Yin, C. Adams, E. Goble, and L. F. V. Madriz, "A classroom at home: children and the lived world of moocs," *Educational Media International*, vol. 52, no. 2, pp. 88–99, 2015.

[12] P. J. Guo, J. Kim, and R. Rubin, "How video production affects student engagement: An empirical study of MOOC videos," in *Proceedings of the First ACM Conference on Learning @ Scale Conference*, ser. L@S '14. New York, NY, USA: ACM, 2014, pp. 41–50.

[13] J. H. Maloney, K. Peppler, Y. Kafai, M. Resnick, and N. Rusk, "Programming by choice: Urban youth learning programming with scratch," in *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '08. ACM, 2008, pp. 367–371.

[14] O. Meerbaum-Salant, M. Armoni, and M. M. Ben-Ari, "Learning Computer Science Concepts with Scratch," in *Proceedings of the Sixth International Workshop on Computing Education Research*, ser. ICER '10. New York, NY, USA: ACM, 2010, pp. 69–76.

[15] D. A. Fields, M. Giang, and Y. Kafai, "Programming in the wild: Trends in youth computational participation in the online scratch community," in *Proceedings of the 9th Workshop in Primary and Secondary Computing Education*, ser. WiPSCE '14. ACM, 2014, pp. 2–11.

[16] N. N. Khairuddin and K. Hashim, "Application of Bloom's taxonomy in software engineering assessments," in *Proceedings of the 8th Conference on Applied Computer Scince*. World Scientific and Engineering Academy and Society (WSEAS), 2008, pp. 66–69.

[17] E. Thompson, A. Luxton-Reilly, J. L. Whalley, M. Hu, and P. Robbins, "Bloom's taxonomy for cs assessment," in *Proceedings of the Tenth Conference on Australasian Computing Education - Volume 78*, ser. ACE '08. Australian Computer Society, Inc., 2008, pp. 155–161.

[18] A. Wilson, T. Hainey, and T. Connolly, "Evaluation of computer games developed by primary school children to gauge understanding of programming concepts," in *European Conference on Games Based Learning*. Academic Conferences International Limited, 2012, p. 549.

[19] 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 *Proceedings of the Second ACM Conference on Learning @ Scale*. ACM, 2015, pp. 131–140.

[20] J. Moreno and G. Robles, "Automatic detection of bad programming habits in scratch: A preliminary study," in *2014 IEEE Frontiers in Education Conference (FIE)*, Oct. 2014, pp. 1–4.

[21] M. Kloft, F. Stiehler, Z. Zheng, and N. Pinkwart, "Predicting MOOC dropout over weeks using machine learning methods," in *Proceedings of the EMNLP 2014 Workshop on Analysis of Large Scale Social Interaction in MOOCs*, 2014, pp. 60–65.

[22] S. Halawa, D. Greene, and J. Mitchell, "Dropout prediction in MOOCs using learner activity features," in *Proceedings of the European MOOC Summit (EMOOCs 2014)*, 2014.

[23] S. Jiang, A. E. Williams, K. Schenke, M. Warschauer, and D. O'Dowd, "Predicting MOOC performance with week 1 behavior," in *Proceedings of the 7th International Conference on Educational Data Mining*, 2014, pp. 273–275.

[24] I. F. de Kereki and V. Pauls, "Sm4t: Scratch MOOC for teens: A pioneer pilot experience in Uruguay," in *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*, Oct 2014, pp. 1–4.

[25] J. Kurhila and A. Vihavainen, "A purposeful MOOC to alleviate insufficient cs education in Finnish schools," *Trans. Comput. Educ.*, vol. 15, no. 2, pp. 10:1–10:18, Apr. 2015.

TUD-SERG-2017-003
ISSN 1872-5392