Introducing Basic Programming to Pre-University Students: A Successful Initiative in Singapore

Heng Ngee Mok, *Member, IEEE*, and Vandana Ramachandra Rao
Singapore Management University, School of Information Systems
mok@ieee.org, vandanarr@smu.edu.sg

***Abstract*—"Let's Code!" is an intensive 3-week basic programming course that aims to formally expose pre-university students in Singapore to programming. This course was conducted in blended-learning format, and included lecture videos, self-check quizzes, video conferences, meet-up tutorials and take-home programming assignments. The authors hope to capture the experience gained from running this course for educators who intend to implement similar courses in the future. Besides a detailed description of this course, significant changes that were made based on feedback from participants and members of the teaching team are documented here.**

Keywords—programming, Curriculum Development, Computer Science Education, Student Outreach, Flipped Classroom, Blended Learning

I. CONTEXT

Programming has surged in perceived "coolness" and popularity in recent years, partly due to the media's portrayal of successful IT companies and well-received outreach programs such as "An Hour of Code" [1] in North America. Since September 2014, the UK's national curriculum included coding lessons for children as young as five: the new curriculum teaches students "how to code, and how to create their own programs; not just how to work a computer…" [2]. Singapore seems to have enjoyed a similar surge in interest in programming with frequent media articles espousing the benefits of learning how to code (for example, see [3]) and the relatively higher starting pay of graduates from computing-related degree programs [4]-[6]. The Singapore government is also very active in equipping citizens with technology skills via its "Smart Nation" initiative [7]. There are various free and commercial offerings of programming (mainly Scratch) courses for kids here [8-10], and school computer clubs commonly introduce programming to members via robots and Internet of Things devices. Programming is a systematic and scientific process that requires the programmer to exercise analytical skills, and the authors believe that learning how to code can improve problem solving ability. The benefits of learning programming is aptly summarized by Steve Job's famous quote: "Everybody… should learn how to program a computer, should learn a computer language, because it teaches you how to think" [11].

Nevertheless the authors noticed that most students in Singapore are not formally exposed to programming. Despite the fact that a few secondary schools are starting to offer "Computing" as an "O"-level[1] subject [12], most government schools do not teach programming, and very few junior colleges (JC) offer "Computing" as an "A"-level subject[2]. Because of this lack of exposure, many JC students may not consider a computing-related course when choosing a degree program at university.

In 2015, the authors obtained a grant to develop and run an intensive 3-week programming course for pre-university students with the main objective of exposing them to programming. The authors wanted to create awareness about this discipline and open up the possibility of a computing-related degree course and a subsequent career in IT. This program was also designed to increase awareness about the degree options offered by the authors' school.

Most of the funding would be used to pay the salaries of mentors (teaching assistants) who were undergraduates majoring in Information Systems at the authors' university. The course would be run four times over two years during the school holidays: in June 2016, December 2016, June 2017 and December 2017. The authors wanted to measure "success" of the project with the following statistics: number of participants successfully trained, the retention rate for the four runs and the responses to the end-of-course survey. This paper provides details about how this course was implemented so that it could serve as a reference for similar initiatives in the future. Feedback collected during the earlier runs were also used to fine-tune how certain things were done. These changes made in the later runs are also documented here.

This project is funded by the MOE (Ministry of Education, Singapore) Academies Fund from 2016 to 2018.

[1] In Singapore, government secondary schools offer four or five-year courses (typically for 13-16 year old students) which lead to the General Certificate of Education (GCE) "Ordinary" or "Normal" level certificate. Most secondary school graduates will apply to study at a junior college or a polytechnic. There are 115 government secondary schools in Singapore in 2016 [13].

[2] Junior colleges offer two or three-year courses (typically for 17-18 year old students) which lead to the GCE "Advanced" level certificate. Most JC graduates will apply to study at university using their "A"-level results.

II. IMPLEMENTATION

A. Administrative Details

The instructors (who are also the authors of this paper) are both experienced programming teachers. New materials were developed and used with existing content to prepare for this programming course. The instructors decided to call this project "Let's Code!", and all participants were called "coders". "Let's Code!" was designed to be a blended course: online learning complimented with face-to-face meet-up sessions. Over the three weeks, there would be seven meet-up sessions: one briefing, five tutorials and one final debrief-cum-examination. In between meet-ups, coders would be engaged with lecture videos and assignments. During the briefing, coders were placed into groups, each headed by a mentor. Coders were able to get help from their respective mentors throughout the three weeks, and mentors were given autonomy over how they interacted with their assigned mentees. To ensure that mentors were fairly compensated, they were paid by the hour, and were allowed to "claim for hours" based on the actual amount of time put in.

In order to encourage coders to complete the course, the authors' school issued Certificates of Completion to coders who met requirements that will be described at the end of this section. To motivate coders to invest more effort and to recognize competency, completers who passed the examination would have their certificates upgraded to a Certificate of Merit.

Coders did not have to pay to attend the course, and could drop out anytime without penalty. Very little marketing was done: the instructors identified a teacher at each JC as the contact person. These teachers helped to advertise the project's official website to their students, who would then register directly there. For all four runs, the instructors took in all eligible applicants. Applicants who turned up for the briefing were officially enrolled as "coders" and given access to the university's learning management system (LMS) portal. Coders were also required to bring along their personal laptops during each meet-up session.

B. Participants' Profile

During registration, applicants were asked if they had ever written a program before (including simple programs), and if so, to list all the programming languages they had used. Over the four runs, only 119 (22.2%) of the enrolled coders listed at least one programming language as their responses to this question. The majority of the coders had never written a program before they started on "Let's Code!". The coders came from a total of 24 pre-university institutions including JCs, secondary schools offering the "Integrated Program" and the NUS High School of Math and Science. 45.4% and 32.3% of the coders were in their second and first year of JC respectively. The remaining were year 4-6 students on the International Baccalaureate program, NUS High School Diploma program, or year 4 Integrated Program students. 76.6% of the coders were Singapore citizens, while 11.8% and 11.4% were Singapore permanent residents and foreigners respectively. 52.2% of the coders were males, and 47.8% were females.

C. Syllabus and Pedagogy

The focus of the course was problem solving, and the syllabus was deliberately designed to include only program control (conditionals, loops), methods and arrays. Object oriented concepts, algorithms and other advanced subjects were excluded. Table 1 shows the topics covered for this course.

TABLE I. SYLLABUS

Unit	*Topics Covered*	*Comments*
0	Setting up the programming environment	To be completed during first meet-up (briefing)
1	Variables, types and basic operations	
2	Decisions and loops	
3	Methods	
4	Arrays	Includes 2D arrays.
5	More problems	Longer and more complicated project-like problems.

Two key points were emphasized throughout the course: (i) there could be multiple "correct" approaches to solve the same problem, and it was worth discussing the advantages and disadvantages of each. (ii) It was not good enough that a method functioned correctly by returning correct values when invoked; it was also important that the code within each method was written efficiently.

The instructors considered Python and Ruby as the teaching language because of their relatively simple syntactic rules (compared to Java). Even though Python had a wider adoption both in academia and industry, Ruby was eventually selected because they had existing teaching materials based on this language.

The instructors were both familiar with the flipped classroom and had experience teaching programming to undergraduates using this pedagogy [14]. For this blended learning course, the instructors decided to employ a similar approach: the more "passive" lectures were to be done at home and delivered in the form of lecture videos, and the limited classroom time during tutorial meet-up sessions would be reserved for active learning activities such as quizzes, in-class programming exercises and code criticism.

Except for unit 0, each of the other five units had accompanying lecture videos, self-check quizzes hosted at the LMS, a tutorial and an assignment. To complete each unit, coders were expected to start by watching the lecture videos and attempting the self-check quizzes that accompanied each video. They were then required to complete and submit the assignment. Each unit concluded with a tutorial meet-up session during which more challenging questions for that unit were worked on in the classroom. Mentors would evaluate the submitted assignment solutions and return a brief report to each coder. Figure 1 summarizes the activities that each coder had to complete for units 1-5.

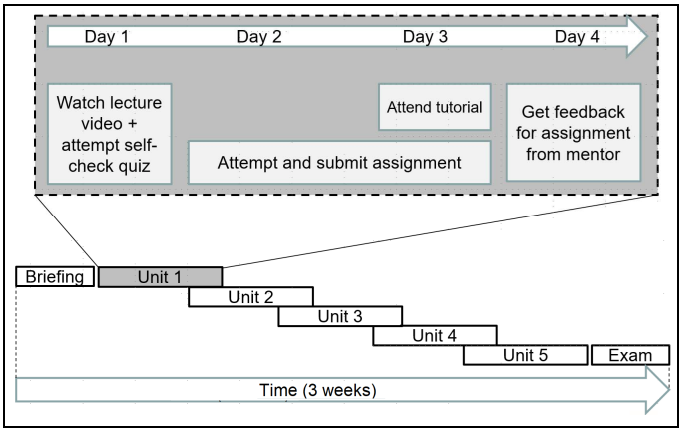

Fig. 1: Sequence of activities for each unit

D. Role of Mentors

Mentors had about one day to prepare formative feedback on the submitted assignment solutions. While that was happening, coders were expected to start watching the lecture videos for the next unit, thus explaining the "overlaps" between the units on the horizontal timeline in figure 1. Mentors were encouraged to key in a summary of their comments into the LMS and correspond with their respective mentees either face-to-face, via online means (WhatsApp, Telegram etc.) or via the telephone to explain their feedback.

Mentors were also specifically instructed not to mark submitted code just by running test cases to validate the values returned; they were expected to "look into" each method to see how the code was written. This was how mentors could add tremendous value over automated code marking systems that merely perform black-box testing on submitted code.

The instructors recognized that the mentors were the first "line of contact" for the coders, and also acted as ambassadors for the authors' school and evangelists for the computing industry; the quality of mentors' interaction would have a significant impact on the coders' overall experience. So the instructors were careful to select suitable mentors. Instead of choosing only from the ranks of the "elite programmers", the instructors preferred mentors who displayed passion in teaching and demonstrated exemplary communication skills.

E. Meet-up Sessions and In-Class Exercises

The first meet-up session was the compulsory briefing. Mentors were present to greet their respective mentees and assist them with connecting to the school's network. The first half of the briefing was spent on administrative tasks: distribution of LMS accounts and a lecture-styled briefing on the pedagogy, schedule and expectations of the course. The second half of the session was spent on unit 0 (Setting up the programming environment). Coders were briefed on the command line environment, how to create and edit a simple Ruby program using a text editor, and how to execute that program using Interactive Ruby. Coders were instructed to leave the lesson only after they had gotten the programming environment installed and properly configured on their personal laptops. At the end of the briefing session, coders should be ready to start working on unit 1.

There were five meet-up tutorials; one for each unit. Each tutorial was three hours long, and was conducted in a 70-seater seminar room. Tutorial sessions were led by the instructors and supported by the mentors. Each instructor ran his/her respective classes slightly differently, but most of the tutorial time was spent working on a set of in-class exercises. Each in-class exercise consisted of ten to 12 questions that could be categorized into three types as shown in table 2. These types of questions reflected the kind of programming skills that the instructors wanted coders to acquire through the program. The assignment questions for units 1-4 were also similarly structured. Mentors were around to assist coders who needed help with these exercises, and each tutorial session ended with a debrief during which solutions from volunteers were flashed on the screen and criticized.

TABLE II. QUESTIONS TYPES IN ASSIGNMENTS AND IN-CLASS EX

	Question Type	***Comments***
1	Theoretical questions	Required coders to read and understand code written by others. By stepping through statements one line at a time, coders appreciated how the program flowed through loops and decision statements, and how variables were updated.
2	Programming questions	Required coders to actually design and write simple code and run them.
3	Troubleshooting questions	Required coders to identify and resolve logical bugs or runtime errors in existing code.

The 7th and last meet-up session included a debrief lecture and an examination. During the course debrief, additional resources were recommended to coders to continue with their learning journey. Code examples written in Python and Java were also shown to impress upon coders that picking up another object-orientated language was not difficult once they had mastered the fundamentals of Ruby. Coders were also introduced to other computing subjects such as artificial intelligence, analytics and cybersecurity as a brief overview of the discipline. The examination was comprised of three parts: an online multiple-choice quiz administered via the LMS, a written theoretical paper, and a hands-on programming test that required coders to write and submit code.

There was one optional lecture conducted via video conferencing between the 4th and last tutorial session. This session was conducted by the instructors using WebEx, and covered the topic of object cloning applied to arrays. The whole conference was recorded and subsequently posted to the LMS for coders who missed the "live" session.

Coders who had fulfilled the following requirements were considered to have completed the course successfully: (i) attended the briefing (first meet-up), the examination (last meet-up) and at least 2 out of 5 of the tutorials, (ii) completed and submitted all five assignments, and (iii) attempted all the self-check quizzes.

III. CHANGES MADE BASED ON FEEDBACK

At the conclusion of each run, the instructors made changes to the next run based on feedback collected from mentors and coders. The significant changes are summarized in this section with justifications for each change.

(1) *Setting the commitment expectations of coders correctly*. During the first run, the instructors received numerous "complaints" about the significant amount of time and effort required for the course during those three weeks. The official course web site was updated to specifically warn potential registrants that they were expected to commit three to four hours per day. Coders were also reminded about the heavy commitment during the briefing session. This resulted in a significant reduction in the number of comments in the end-of-course survey about the course requiring too much time and effort.

(2) *Allocation of two mentors to coders*. For the first three runs, one mentor was assigned to a group of 10-15 coders. In the fourth run, the instructors decided to assign two mentors to a larger group of 20-30 coders. Mentors decided how work was to be split between the pair. Whilst the mentor/coder ratio was kept the same, the instructors felt that this was a preferable arrangement for the following reasons:

- a less experienced mentor could be paired with one who was involved in an earlier run,
- mentors could learn from each other,
- coders had a choice as to which of the two assigned mentors to speak to,
- mentors were able to more easily take over his or her partner's duties when necessary, and
- there was more interaction and cooperation amongst mentors.

(3) *Providing screencasts showing worked examples of assignment questions*. Especially for programming, there is a gap between understanding the programming concepts in theory, and actually coding and running a functioning program. There were comments such as "I could understand everything in the lecture videos and was able to get a perfect score for the self-check quizzes, but was unable to start on the (programming) assignment". From the second run onwards, about four out of 12 of each assignment's questions were converted into "example questions", and screencasts on how to tackle them were released together with the assignment. These videos covered alternative approaches to solve the same question with comments on the advantages and disadvantages of each. Coders were strongly encouraged to watch them even if they managed to derive a working solution to these example questions without any help.

(4) *Putting friends together in the same teams*. The instructors were aware of the benefits of heterogeneous teams over homogeneous ones [15]. However, for this course, the instructors deliberately placed coders from the same schools into the same groups whenever possible, under the assumption that team members were more likely to know one another, and hence would "push" one another along to completion. From the third run onwards, the course web site was edited to enable registrants to "register together" with friends so that they would be placed into the same group even if they were from a different school. The instructors suspected that it was more likely for "loners" to drop out of the course, than for coders who were working on the assignments together.

(5) *Having tutorials before assignments*: For the first three runs, coders were instructed to follow this sequence for each unit: (i) watch lecture videos and attempt self-check quizzes, (ii) attempt assignment, (iii) attend tutorial. Tutorial sessions were supposed to "wrap up" a unit, and the in-class exercise questions given out during tutorials were slightly more challenging than the assignment questions. However the instructors realized quickly that there were coders who turned up for tutorial sessions without having even watched the lecture videos for that unit. Mentors were also giving feedback that some coders still had difficulty bridging the gap between the lecture videos and the programming assignments even after watching the screencasts of "example questions". For the fourth run, the instructors reshuffled the sequence to: (i) watch lecture videos and attempt self-check quizzes, (ii) attend tutorial, (iii) attempt assignment. This time round, the assignment questions were slightly more challenging than the tutorial questions. Figure 2 shows the change. This new arrangement seemed to facilitate learning better: coders who turned up at the tutorials were able to get some guided practice before they started on the assignments on their own, and mentors had one more day to motivate coders who turned up for tutorials without having watched the lecture videos.

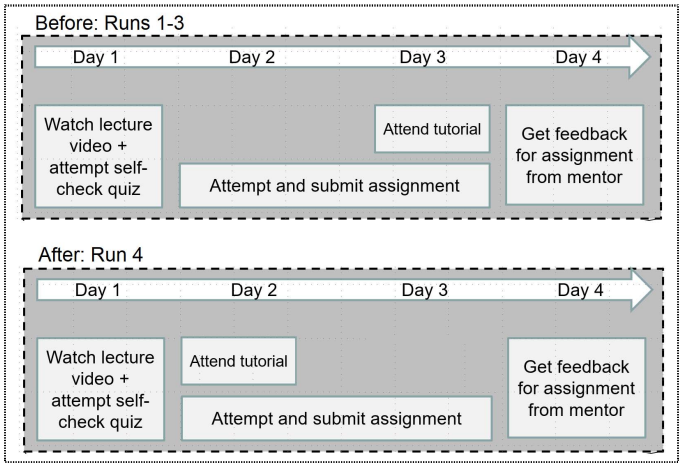

Fig. 2: The tutorial was shifted from day 3 to day 2 for each unit in the last run

(6) *Including an "open-ended" project in the last assignment motivates interested coders to self-explore*. For the fourth run, the instructors decided to include an "open-ended" project-like question in unit five's assignment. Basically coders could select an existing problem or game and write a program to solve it. To encourage creativity and effort, a small percentage of their exam scores was allocated to this project. This change motivated proficient coders to "try new things" and be rewarded for exploration. The instructors received some excellent submissions for this project that were very impressively coded.

(7) *Using a "dashboard" to track the progress of individual coders*. "Let's Code!" was a very fast-paced course with scheduled tasks to complete every day. Some of the coders who did not complete the course gave up because they allowed undone work to snowball, and dropped too far behind schedule until a sense of helplessness overwhelmed them. It was useful for the instructors to have a "bird's-eye view" of the general progress of all coders, and yet be able to identify potential drop-outs, so that more attention could be given to them. Most LMSs come with dashboards that provide instructors with such insights, but they are expensive to customize. For the last two runs of "Let's Code!", the instructors prepared an online spreadsheet with columns for attendance, submission dates of each assignment and scores for the self-check quizzes. Mentors were instructed to update the spreadsheet once they had given feedback for assignment submissions. Attendance records on the same spreadsheet were also updated at the end of each meet-up. This online spreadsheet – although primitive compared to the more fanciful ones which come with graphs and charts – gave the instructors a good overview of "what's happening" as the course progressed. It also alerted mentors to lagging mentees who were likely to drop out unless more attention was given.

IV. Course Statistics

A total of 535 coders were enrolled over the four runs, 80.6% of whom completed the course and were awarded certificates. The attrition rate of 19.4% was considered low because there was really no obligation for coders to complete the course. The common reasons given by coders who withdrew officially were the huge amount of time required by the course, co-curricular activity commitments during the holiday period, the need to prepare for mid-year examinations, need to go overseas, loss of interest, and inability to follow the lessons.

The instructors pegged the final examination at roughly the same level as a first programming course taken by freshmen at the authors' university, and were confident that coders who passed the examination could be considered competent beginning programmers. As stated earlier, completers who passed the examination had their Certificates of Completion upgraded to Certificates of Merit. 50.6% and 49.4% of the completers were awarded merit and completion certificates respectively.

Even though the course was tailored for pre-university students who had never written code before, the instructors were expecting coders who had prior programming experience to do better. Of the 119 coders who had written a program prior to the course, 59.7% and 25.2% of them were awarded merit and completion certificates respectively. In comparison, 35.1% and 43.8% of the coders without prior programming experience were awarded merit and completion certificates respectively.

Coders present for the final meet-up session were invited to fill up the end-of-course feedback survey before they sat for the examination. For the last two runs, the authors employed the FACET (For Assessment of Continuing Excellence in Teaching) instrument developed and used by the authors' university [16] with a few customized questions inserted. Course feedback were very positive. On a scale of 1 (for "Extremely poor") to 7 (for "Excellent"), respondents gave average scores of 6.3/7 for both "Overall rating of instructor" and "Overall rating of mentor" (n=228, s.d.=0.8, response rate=97%). The "Learning experience in this course", "Quality and value of course material" and "Quality and usefulness of course assignments/projects" each scored a relatively high 6.1/7.

In order to determine if the course had affected the likelihood of coders pursuing computing in the future, respondents were asked to rate the following two statements on a 7-level Likert scale (Strongly Agree to Strongly Disagree):

- Q1: "Before this course, I have considered a career or university program in an IT/computing/computing-related discipline".
- Q2: "After this course, I will consider a career or university program in an IT/computing/computing-related discipline".

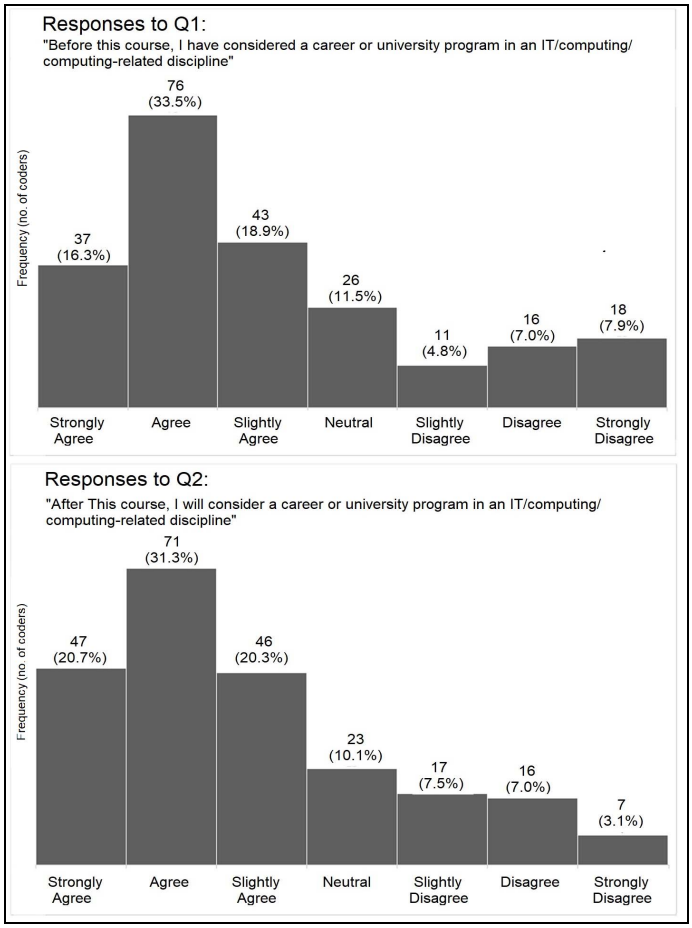

Fig. 3: Histograms showing distribution of responses to Q1 and Q2.

Figure 3 shows the responses to both questions (n=227). The graphs seem to imply that several respondents who initially had not considered a future in computing had changed their minds. Responses were mapped to a numeric value ("Strongly Agree"=7, "Agree"=6…"Strongly Disagree"=1), and compared for each respondent. 34.8% of respondents gave a higher score

for Q2 than Q1, signalling that the course had a positive effect in encouraging them to consider this discipline. 17.2% of respondents gave a lower score; these were likely to be participants who realised that they did not enjoy programming after all. 48.0% of the participants gave identical responses for both questions. Despite these positive figures, it has to be noted that the survey was done only by coders who attended the last meet-up session.

Coders were specifically briefed during the first meet-up session that in order to qualify for a certificate, they had to be present at the final meet-up session for their examination, and at least 2 out of 5 of the tutorial sessions. 76.3% of the coders, however, attended more than two tutorial sessions, with the majority (34.4%) having attended all five sessions. Figure 4 shows how many tutorial sessions coders attended (n=535). The 18 coders who attended none of the tutorials were those who dropped out after the briefing.

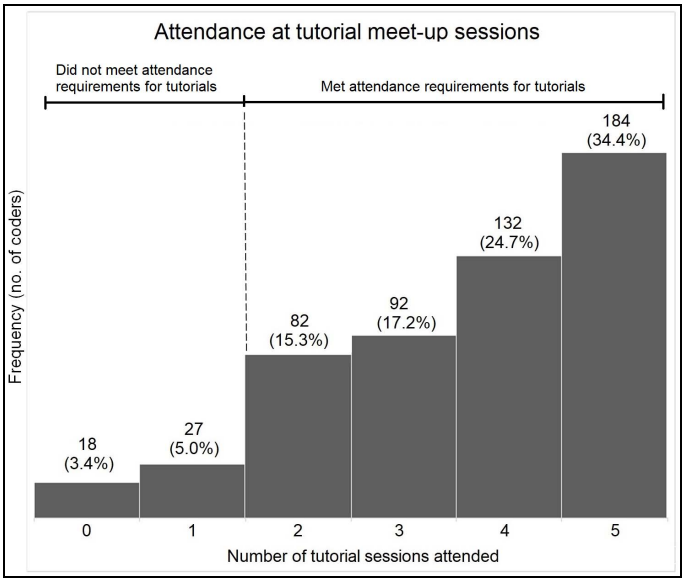

Fig. 4: Histogram showing the distribution of number of tutorial sessions attended by coders.

There seems to be a correlation between the number of tutorial sessions attended by coders, and the category of certificates they obtained eventually. The average attendance of merit certificate holders, completion certificate holders and those who did not finish the course were 4.1, 3.8 and 2.0 respectively (out of 5 tutorial sessions).

Coders were also asked to indicate the number of hours spent on this course per day (including attending meet-up sessions) over those three weeks. Respondents gave an average figure of 3.4 hours per day (n=228).

In order to determine the impact of this course on school marketing, respondents were also asked if they were "aware that there is a School of Information Systems" at the authors' university prior to attending "Let's Code!". 55.6% (n=522) of the coders replied "not aware", thus underscoring the role of this project in marketing the school and its degree courses to potential students.

V. Concluding Remarks

The authors consider "Let's Code!" a successful program: over the four runs, a total of 535 coders enrolled in the course, and 431 (80.6%) of them completed it. End-of-course ratings for instructors, mentors and the course were very high, and there were many heart-warming notes from coders on how participation in this program had changed perspectives on programming and computing. Some had discovered a new interest and would be matriculating in computing-related programs for their university studies. Despite the requirement to attend only 2 out of 5 tutorial sessions, the course enjoyed high attendance rates with more than one-third being present for all five sessions. The authors were glad that the 412 participants who had never written a line of code before, wrote their "Hello World!" program through this project. 78.9% of these first-time coders managed to complete the course, and more than one-third of them gained enough skills over the three weeks to be recognized as competent beginning programmers by the authors. The project also contributed tremendously to school outreach and marketing: through "Let's Code!", more than half of the participants became aware of the authors' school and the degree options that could be pursued there.

The authors were able to fine tune the pedagogy by studying feedback from coders and mentors after each run, and implement constructive changes in the following run. Undergraduate students who were involved as mentors also gained teaching experience, leadership and coaching skills.

This project had made an impact on a number of pre-university students in Singapore by introducing programming to them. This could have opened up an entire new world of educational and career possibilities.

Acknowledgment

The authors would like to acknowledge their collaborators and colleagues: Eileen Moh and Tan Ai Chin from Hwa Chong Institution, Prof. Steven Miller, Prof. Venky Shankararaman, Koh Kwan Chin, Fiona Lee and the General Office staff from Singapore Management University, as well as the following organizations which supported this initiative: Science Centre Singapore, IEEE (Singapore branch), Singapore Computer Society and the Infocomm Media and Development Authority.

References

[1] Code.org, Hour of Code. [Online]. Available: https://code.org/about [Accessed Apr. 21 2018].

[2] S. Dredge, "Coding at school: A parent's guide to England's new computing curriculum", *The Guardian*, Sep. 4, 2014. [Online]. Available: http://www.theguardian.com/technology/2014/sep/04/coding-school-computing-children-programming [Accessed Apr. 21 2018].

[3] H. W. Lauw, "Commentary: The hidden value of learning how to code", *Channel NewsAsia*, Feb. 18, 2018. [Online]. Available: http://www.channelnewsasia.com/news/singapore/commentary-coding-programming-enrichment-classes-children-value-9905320 [Accessed Apr. 21 2018].

[4] R. Y. K. Toh, "Fresh grads in computing see biggest salary jump", *The Straits Times*, Feb. 23, 2017. [Online]. Available: http://www.straitstimes.com/singapore/education/those-in-computing-see-biggest-starting-salary-jump [Accessed Apr. 21 2018].

[5] P. T. Wong, "Move over doctors and lawyers, there's a new rich kid in town", *Today*, Feb. 26, 2018. [Online]. Available:

http://www.todayonline.com/singapore/move-over-doctors-and-lawyers-theres-new-rich-kid-town [Accessed Apr. 21 2018].

[6] E. Leung, and S. Tan, "IT, business graduates get higher starting pay", *The New Paper*, Feb. 27, 2018. [Online]. Available: http://www.tnp.sg/news/singapore/it-business-graduates-get-higher-starting-pay [Accessed Apr. 21 2018].

[7] Smart Nation and Digital Government Office, Smart Nation. [Online]. Available: http://www.smartnation.sg/about/Smart-Nation [Accessed Apr. 21 2018].

[8] L. K. Fai, "Google programme inspires kids to learn coding", *Channel NewsAsia*, May 20, 2017. [Online]. Available: http://www.channelnewsasia.com/news/singapore/google-programme-inspires-kids-to-learn-coding-8868844 [Accessed Apr. 21 2018].

[9] K. Kwang, "Speaking in code: Kids in Singapore are learning a new language", *Channel NewsAsia*, Mar. 16, 2017. [Online]. Available: http://www.channelnewsasia.com/news/singapore/speaking-in-code-kids-in-singapore-are-learning-a-new-language-8188110 [Accessed Apr. 21 2018].

[10] L. Hio, "Coding classes for kids in high demand", *The Straits Times*, Mar. 8, 2016. [Online]. Available: http://www.straitstimes.com/tech/coding-classes-for-kids-in-high-demand [Accessed Apr. 21 2018].

[11] M. Rosoff, "The most interesting thing Steve Jobs said in a 'lost' interview showing next week", *Business Insider*, Nov. 11, 2011. [Online]. Available: http://www.businessinsider.com/the-best-quotes-from-the-lost-steve-jobs-interview-showing-this-weekend-2011-11 [Accessed Apr. 21 2018].

[12] P. Lee, "19 Schools to offer programming at O levels", *The Straits Times*, Feb. 29, 2016. [Online]. Available: http://www.straitstimes.com/singapore/education/19-schools-to-offer-programming-at-o-levels [Accessed Apr. 21 2018].

[13] Singapore. Ministry of Education, *Education Statistics Digest*. p.2. Singapore: Ministry of Education; 2017. [Online]. Available: https://www.moe.gov.sg/docs/default-source/document/publications/education-statistics-digest/esd_2017.pdf [Accessed Apr. 21 2018].

[14] H. N. Mok, "Teaching tip: The flipped classroom", *Journal of Information Systems Education*, vol. 25, no. 1, pp. 7-11, Spring 2014.

[15] S. Kagan, "10 Reasons to Use Heterogeneous Teams", *Kagan Online Magazine*, Fall 2014/Winter 2015. [Online]. Available: http://www.kaganonline.com/free_articles/dr_spencer_kagan/396/10-Reasons-to-Use-Heterogeneous-Teams [Accessed Apr. 21 2018].

[16] FACET (For Assessment of Continuing Excellence in Teaching), Singapore Management University, 2012. Available: http://cte.smu.edu.sg/feedback-teaching/student-feedback-teaching/resources [Accessed Apr. 21 2018].

Heng Ngee Mok and **Vandana R. Rao** are both with the School of Information Systems, Singapore Management University. Mok teaches concurrent programming and computational thinking. Vandana teaches programming and data management.