Cogent Education

ISSN: (Print) (Online) Journal homepage: https://www.tandfonline.com/loi/oaed20

Types and time of interaction for teaching introductory programming using instruction method of extreme apprenticeship

Srinivasan Lakshminarayanan & N. J. Rao |

To cite this article: Srinivasan Lakshminarayanan & N. J. Rao | (2021) Types and time of interaction for teaching introductory programming using instruction method of extreme apprenticeship, Cogent Education, 8:1, 1969880, DOI: 10.1080/2331186X.2021.1969880

To link to this article: https://doi.org/10.1080/2331186X.2021.1969880

Published online: 04 Sep 2021.

Lakshminarayanan & Rao, *Cogent Education* (2021), 8: 1969880
https://doi.org/10.1080/2331186X.2021.1969880

cogent education

FOOD SCIENCE & TECHNOLOGY | RESEARCH ARTICLE

Types and time of interaction for teaching introductory programming using instruction method of extreme apprenticeship

Srinivasan Lakshminarayanan[1]* and N. J. Rao[2]

Received: 21 December 2020
Accepted: 13 August 2021

*Corresponding author: Srinivasan Lakshminarayanan, Department of Computer Science and Engineering, Jain (Deemed to Be University), Jakkasandra, Ramanagara, Karnataka, India
E-mail: srinirad@gmail.com

Reviewing editor:
Luís Tinoca, Universidade de Lisboa, PORTUGAL

Additional information is available at the end of the article

Abstract: CS1 courses are designed in Indian Institutions as a lecture course of three to four credits and one credit lab course. The issues related to curriculum design, instruction design, and students' learning manifest themselves as issues in the lab programs. This situation presents the lab instructor with an opportunity to understand and address the difficulty the student is facing. The difficulty could be understanding a concept, applying a concept, the amount of effort invested, and the time required by the individual to solve the problem. The student might need help to address various emotional aspects related to peer pressure, need for completion, need for acceptance by the instructor, and achievement goals. The student's difficulty is usually handled by a) allowing the student to correct programs by looking at the working programs of their peers or class notes, b) The teacher or peer fixes the program. The problem gets solved, and the student moves on to the next program, but the student's underlying difficulty may not have been resolved. Since addressing the underlying difficulty takes more time, we offered the students a voluntary supplementary CS0 course using the Extreme Apprenticeship instruction method. In this study, we estimated that students need between four to fifteen hours of one-on-one synchronous interaction time with the instructor based on

ABOUT THE AUTHOR

Srinivasan Laksminarayanan worked as a Software Engineer in various roles and is currently pursuing a Ph.D. at Jain University, India. He held the position of Development Manager at Oracle (India) before switching to education research.
N. J. RAO received Ph.D. in Control Systems from the Indian Institute of Technology, Kanpur. He was the Chairperson of the Center for Electronics Design and Technology and subsequently the Chairperson of the Department of Management Studies in the Indian Institute of Science. His research interests included Control Systems, System Dynamics, and Higher Education. Since superannuation in 2006, he is working in several areas related to Higher Education in India. His current research interests include assessment, accreditation, and metacognition in Higher Education.
This study is a part of ongoing research to achieve transformative learning in introductory programming courses.

PUBLIC INTEREST STATEMENT

Every year close to 700 thousand students attend introductory programming course as a part of the first year of the undergraduate engineering programs in India. The lecture based instruction method is predominantly used in most institutions. These instruction methods result in students learning programs instead of learning programming. In order to address the situation we adopted the Extreme Apprenticeship method. Any change in the instruction method will have an impact on the instructor's effort. This study measures one on one instructor interaction time when the Extreme Apprenticeship method of instruction is adopted. Experienced teachers over years of interaction develop specific communication patterns. In this study the interactions between an instructor and students were captured and analysed to identify themes of interaction. These themes can be used to train teachers who are new to teaching introductory programming courses using the Extreme Apprenticeship method.

prior exposure. Thematic Analysis of interactions identified fifteen themes in metacognitive domain interactions, eight themes in cognitive domain interactions, and six themes in affective domain interactions.

Subjects: Computer Science (General); Engineering Education; Teaching & Learning

Keywords: CS1; introductory programming course; cognitive apprenticeship; extreme apprenticeship; metacognitive; affective

1. Introduction

Cognitive apprenticeship is an instruction method where the instruction is delivered as modelling, coaching, scaffolding, articulation, reflection and exploration (Collins, 1991). Modelling is the demonstration of performing a task while articulating the rationale behind each action and decision taken by the instructor. Coaching by the instructor is done by observing and facilitating student's learning by the instructor while the student is performing the task. Scaffolding is a way of organizing the content by breaking down the task to smaller tasks or tasks created to help the student learn concepts needed for performing a task. Articulation refers to the students making their thinking processes explicit. Reflection refers to the students comparing their performance with that of peers and the instructor and thereby developing the ability to perform. Exploration happens when the students do not need the scaffolding by the instructor, and the instructor predominantly performs the role of evaluation of the work Figure 1.

The basic requirements for cognitive apprenticeship are instructor's expertise in the subject and ability to design and deliver the course using the cognitive apprenticeship method. Instructor time required should be allotted by the curriculum and the institution. The students should have mastery goal orientation towards the course. About 800 students attend the CS1 course every semester in the institution where the study was done. It is a mandatory course for all engineering undergraduates. The study was conducted in the year 2020, January to June semester batch. C was the programming language used for the course. The lab instructor (LI) in the study was assigned 164 students in 12 batches. The students had to solve about three problems in every 2-hour lab session over a period of 14 weeks. Majority of the students could not complete even one lab program. Hence the students would type out the programs looking at the class notes or reference programs, and the errors were solved by comparing with the class notes or peers' programs. When the student fails to write the program by comparing the code, the instructor helps the student. Though the current system students learn the existing programs, many students found the programming very difficult because of the syntax, and logical errors that inad-

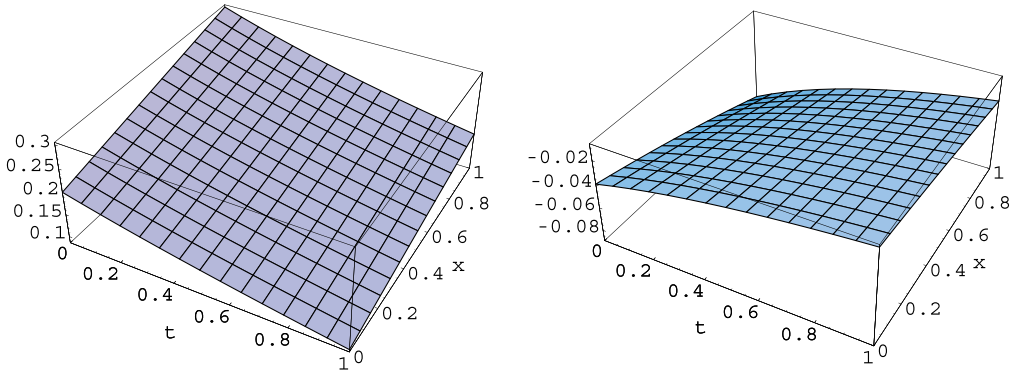

Figure 1. Example of a two-part figure with individual sub-captions showing that captions are flush left and justified if greater than one line of text.

(a) An example of an individual figure sub-caption.

(b) A slightly shorter sub-caption.

vertently happen while typing. The LI proposed to offer a supplementary course (CS0) based on cognitive apprenticeship method to address the difficulties of the students.

The CS0 course had the following characteristics:

(1) It was not graded.
(2) The institution offered a certificate to all the students who completed the course.
(3) It was optional and voluntary.
(4) The instructor's availability online and in the lab was high.
(5) The course outcome was that students should be able to write a program for an authentic programming task.
(6) The instruction was one to one.

The version control data and interactions with the instructor form a basis for understanding the individual student's learning. The students who do not perceive any intrinsic value had a choice to drop from the course without consequences to their grades. Twelve students underwent the majority of the course with the interactions exclusively on the text chat. Teaching on text chat makes teaching a linear process and captures all the teacher-student interactions. This study intends to analyze the course's chat logs to achieve the following research objectives:

(1) Identify metacognitive, cognitive and affective interactions that facilitate first-year students learning the basics of programming.
(2) Estimate interaction time needed between a student and the teacher for a novice to learn programming basics.

2. Related work

Ureel and Wallace (2018) report in their study that cognitive apprenticeship results in an improvement in student performance and a clearer understanding of the place of communication in the lives of computing professionals. Brondino et al. (2019) reported that higher compliance of students in an extreme apprenticeship course performed better and was characterized by less intense anxiety, anger, and hopelessness compared to those with lower compliance. Vihavainen et al. (2011) in their work describe extreme apprenticeship as an extension of Cognitive apprenticeship. We adopted the extreme apprenticeship method for this study. We adopted, from their work, the values of learning by doing, continuous feedback, no compromise, and the student should be able to write programs for an authentic task. We followed the practices of minimal lecturing and encouraged looking for information. The students were predominantly taught online using online google hangouts, and the activities were coordinated using google classroom and online git version control system.

Haatainen et al. (2013) have described the implementation of additional support for students finding the CS1 course difficult. In the study, they used self-assessment to identify students needing help. They created a learning environment where participation was voluntary. The students who participated found lower social barriers to learning since they met others having similar difficulties. The learning was done by programming in groups before graduating to programming individually. This process allowed students to learn programming in a socially supported environment. They recommend that differentiated learning and educational constructionism will benefit students learning. We have adapted some of the methods from this study to address the issues in our instructional setting. We reduced the social barriers by not having grading and making the course a voluntary mastery learning course. The student asking questions was a mandatory beginning to every interaction, and this reduced the social barriers further. Interventions were attempted in the order of pure discovery, guided discovery, direct questioning, direct answering, and direct instruction.

Campbell et al. (2019) implemented a self-paced mastery learning course online and reported that though the student reported a deeper engagement, the student tended to procrastinate work till the end of the course. In our course, we observe that procrastinating students drop out eventually. McCane et al. (2017) reported the advantages of a mastery learning course in programming for weaker students. They reported the difficulties in the implementation of the mastery course. We feel that trying to run a mastery course in a semester format leads to these problems. We ran the course as a self-paced but regular number of hours every week in mastery learning mode without any grades as a voluntary supplementary course, which allows the course to focus on students ability to write a program for authentic tasks instead of completion within a given time period. Student's intention to learn becomes a necessary condition for the continuation of the student course.

Xie et al. (2019) proposed a theory in which they identify four distinct skills that novices learn incrementally. These skills are tracing, writing syntax, comprehending templates (reusable abstractions of programming knowledge), and writing code with templates. They mention that incorporating the theory in the instruction design resulted in improved instruction. Prather et al. (2019) identified that students have metacognitive difficulties due to forming the wrong conceptual model about the problem, dislodging an incorrect conceptual model of the problem, assuming the correct conceptual model for the wrong problem, and moving too quickly through one or more stages incorrectly leads to a false sense of accomplishment. We find that cognitive apprenticeship addresses the methods and issues reported in the studies. Tracing and writing syntax is addressed in almost all courses as a norm. We incorporated comprehending templates and writing code with templates as steps in the code review process of the CS0 course. In addition, the metacognitive difficulties mentioned were addressed during the continuous feedback adopted in our study.

Mayer (2004) described the disadvantages of pure discovery learning and recommended guided discovery. We have taken the position that the student's cognitive load decreases with the increased familiarity of content over time. By carefully managing the time for pure discovery, we can give the student the experience of pure discovery learning. This course aims to provide students with the experiences of pure discovery, that is, the ability to arrive at the right solution using the internet resources and guided discovery where the instructor attempts corrective feedback before attempting explanatory feedback. The student can choose to learn the concepts from many available resources online, but they will have to write the program by themselves when it comes to solving an authentic task. If they cannot solve and if the instructor has to give direct instruction, then the student will have to solve another authentic task. This cycle continues until the student can write a program for an authentic task.

Raj et al. (2018) adopted live-coding as an instruction method. This is a practice where the instructor writes code in the class while talking out loud, explaining their thought process. They claim this is a useful pedagogical tool to learn programming. Students directly get to learn algorithmic thinking, debugging skills and other programming practices from a master. This is a direct implementation of cognitive apprenticeship in an introductory programming course. Since our instruction was individually paced, we preferred explaining an already written program and the rationale behind the decisions for writing the code in a specific way.

Rodríguez-Bonces and Ortiz (2016) conclude that in their programming course, the methods of the cognitive apprenticeship enhance online collaborative learning not only because students work together to reach a common goal, but also because they can support each other's learning through synchronous interactions when using a chatroom for this purpose. In our study, we use the synchronous interaction aspect to maintain the continuity of learning between the instructor and the student.

Recent research indicates a tilt toward automation. Automated grading, automated feedback, online tutorial, MOOCs, educational analytics have all been used to improve the learning of the students. Increasingly students are using these systems to learn. These systems do not generally support code review, and the students find them excessively strict (Wilcox, 2015). Yan et al. (2019) implemented a feedback tool as a way to begin the conversation on metacognition and as a way to bring a human aspect back to programming despite the size of classrooms. In this study, the students were required to use a version control system, and the data from the version control system was used for feedback. Students were encouraged to use any online resources, and the instructor intervenes when the student is not able to progress in an activity or seeks help.

3. Instruction design

Table 1 lists the activities and corresponding cognitive outcomes. The design was arrived at based on the experiences with previous batches in previous years. The course is a supplementary course, and the students also attend a regular CS1 course. Right from the first activity, the instructional method starts with discovery learning methods, followed by guided discovery learning methods. When both the methods do not lead to progress, and the student still has the difficulty, then the direct instruction method is used. Since this process involves elaborate work, the concepts taught were limited to those required for writing programs for chosen authentic tasks. The authentic tasks were chosen from ACM high school programming contests.

4. Method

Mixed methods were adopted for the study. The participants interacted with the instructor exclusively using the text chat. The programs were submitted for review using the git revision control system. Quantitative methods were used to analyze the count data from chat logs and estimate the time taken for interaction between the instructor and the student. The synchronous interaction time is the sum of all time differences between two messages when the difference is less than two minutes. Qualitative thematic analysis method with manual coding was used to classify the chat interactions and understand the kinds of interactions used to resolve the students' difficulties. Only the participants who interacted extensively using chat were selected for the analysis.

Table 1. CS0 activities

Activity	Cognitive Outcomes
Write a hello world program and write 20 questions on each aspect of the program.	Use the development environment. Understand questioning.
Write a program to add two numbers.	Understand data types and expressions.
Write a program to add two numbers using four functions.	Understand modularization in C.
Write a program to find the distance between two points.	Apply modularization in C.
Write a Program to find the distance between two points using structures.	Understand data abstraction in C.
Write a program to find the sum of n different numbers.	Apply for loops and arrays to handle process data of many objects of the same kind in C.
Write a program to find the sum of two fractions.	Apply modularization and data abstraction in C.
Write a program to add n fractions.	Apply modularization, data abstraction to handle data of many objects of the same kind in C.
Write a program for the task from ACM High School Programming Competition with assistance.	Write a program for an authentic task with instructor assistance.
Write a program for the task from ACM High School Programming Competition without assistance.	Write a program for an authentic task without instructor assistance.

Table 2. Quantitative summary of student interactions

Student	No. of Msgs by Student	No. of Msgs by Instructor	Problems Completed	Synchronous Interaction time	Prior Exposure	Git Commits	Software Review Requests
N	266	387	10	3:57:09	No	41	27
U	275	342	7	9:43:33	No	46	50
A	981	1018	10	6:39:38	No	73	27
P	511	632	7	10:52:07	No	36	28
SH	852	936	7	12:10:24	No	85	48
SI	183	394	6	8:22:39	No	37	8
Y	308	473	10	3:58:58	Yes	60	35
J	512	712	6	4:45:46	No	75	46
SU	433	375	10	3:31:06	Yes	50	40
ST	1257	1200	10	5:16:52	Yes	86	46
KV	124	142	10	5:20:25	Yes	50	14
VA	356	391	8	1:48:41	Yes	54	51

5. Quantitative analysis of data

From Table 2, we can see that only 50% of students were able to write a program for an authentic task even with additional instruction. Students, who had prior exposure, needed instruction time of 2 to 5 hours to be able to write a program for an authentic task. We see from the table that students who do not have prior exposure require between 4 to 15 hours of interaction time. This data can be used to determine the number of credits to the lab course and the number of students assigned to an instructor. Even if we assume 6 hours of interaction time per semester per student on an average, one-credit lab course, which is 2 hours per week, and a 14-week course, a lab instructor can handle a maximum of five students. In the institution where the study was conducted, each instructor handles 12 students, which reduces the interaction time to less than 3 hours per student per semester, indicating the need for the management to rethink the curriculum design for the course.

6. Qualitative data analysis

Ben-Ari (2001) writes that a researcher working from a constructivist viewpoint should use qualitative methods. The insights obtained from qualitative research are far more helpful than the research that measures performance alone and then draws conclusions on the success of a technique. Magrini (2012) defines phenomenology as a philosophical "method/practice" of observing, recording, and interpreting "lived experience" through vivid and detailed descriptions. The practice of phenomenology seeks to expose, uncover, or reveal "universal" (transcendental) elements of human existence that are instantiated within practical, "particular" empirical situations. In this study, the instructor interacted with the students completely on the text chat for synchronous communication. Each student had between 400 to 2000 interactions with the instructor and hence capturing the lived learning experience in great detail.

Instructors encounter many contexts repeatedly during the interactions in the extreme apprenticeship method in introductory programming courses. Since this course happened exclusively on the text, the chat logs provide rich content for identifying these repeated contexts and also a chance to reflect upon the way the interaction happened for a given context. In this study, we followed the coding stages of reflexive thematic analysis for identifying the contexts and classifying the interactions. We went through multiple rounds of coding to identify the codes since we had to code each student resulting in refining the codes and combining the codes. We then classified code into metacognitive, cognitive and affective domains. The whole process intends to develop a repeatable reflexive practice. The thematic analysis followed the steps of familiarization, coding and generation of initial themes by the lab instructor and was refined further when the chat logs of the next student were coded. We present our understanding based on the themes identified during the coding process.

7. Interactions with the students in metacognitive domain

Table 3 is a summary of codes for interactions related to the metacognitive domain. The most important cultural shift was moving the student from Understand to Apply (TUVSCONS). From the prior educational experiences, many of the students believe that understanding a solution explained by the teacher is the end of learning. Programming falls under apply, analyze, evaluate, and create cognitive levels Sorva. TUVSCONS interaction happens when a student requests for a direct answer from the instructor. The instructors get an opportunity to present the constructivist methods of learning. From the data, we see 10 out of 12 students had this interaction with the instructor. The time required to write a program is significantly more than understanding and reproducing the program. This shift leads to many metacognitive, affective, and cognitive domain interactions with the teacher. The student will now have to plan their learning to accommodate this additional time requirement from their study time. Table 4 shows chat excerpts related to understanding versus constructing programs.

A common question that comes to a novice programmer's mind is when a program is working, why should they improve it. They do not see the need to modularize, making the programming more readable, using appropriate data types, indenting the program, or using appropriate loops. One of the

Table 3. Metacognitive interaction with students

Code	Metacognitive Interaction	Number of Interactions	Number of Students
TUVSCONS	Student made aware of the difference between understanding programs and apply.	22	10
TTIME	Student should give time to themselves.	36	9
TWCVSWWC	Student made aware of the difference between working code and well-written code.	11	8
TFOCUS	Student should focus on the current difficulty.	8	5
TLGVSLL	Student should be aware of the differences between learning for grades versus learning programming.	5	4
TMDEBUG	Student and teacher reflect on strategy.	7	3
TPQVSPA	Helping peer by questioning vs helping peer by answering.	6	5
TLISTENING	Student having difficulty receiving.	1	1
TOBEYVSAGREE	Student obeying without questioning.	1	1
TPLAN	Student and teacher plan	1	1
TRCN	Student reminded to refer class notes while solving.	32	11
TREPEAT	Student asked to repeat to improve understanding.	12	5
TRIN	Student asked to read internet reference to improve understanding.	6	6

major difficulties lies in communicating with the student that the program is wrong even though it works. Students cannot take the comments as objective comments and need discussion to get the correct attitude towards code reviews. Whenever an attitude needs to be fixed, it takes additional interactions and time. Student learning time is extremely precious. They have difficulty understanding that spending time refining a working solution is not a waste of time and is essential. Table 5 shows chat excerpts related to working programs versus well-written programs.

Interactions were required to delineate learning for grades and learning to program (TLGVSLL). Interactions happened with 4 out of twelve students. When the students could not progress as fast as they would like to, they needed counselling to allot more time (TTIME) and improve their learning strategies (TMDEBUG). 9 out of 12 students needed interactions with respect to giving more time to the problem, and three students needed interactions for improving learning strategies. The students needed to be counselled on peer helping (TPQVSPA) since there was a general tendency to share the program once they got it working. When there was a lack of

Table 4. Chat excerpts related to understanding vs constructing programs

LI: Still, higher levels of construction needs to be achieved.	LI: If I debug, how will it help you? It is not a conceptual question. it is just a matter of staying on the problem and debugging it.
STUDENT: Okie.	STUDENT: Well, sir, I couldn't do it.
LI: Arriving at using arrays to return multiple values from a function was known but was not applied without prompting.	LI: You didn't do it, that you couldn't is a false statement.
STUDENT: Okay.	STUDENT: Couldn't, sir.
LI: Copying before modification was known but not applied.	LI: Didn't.
STUDENT: I didn't get that idea.	STUDENT: I did try, sir.
LI: There were two new things you understood about them but not constructed … Do you see that in life, you get only one opportunity to construct?	LI: Didn't try a sufficient length of time.
STUDENT: Yes sir.	STUDENT: One day, sir
LI: Once you have understood, the opportunity to construct is gone forever and forever, but there are infinite such opportunities, and you have the potential.	LI: Next time, two days before giving up.
STUDENT: Yes, sir.	STUDENT: Okay, sir.

Table 5. Chat excerpts related to working programs vs well written programs

LI: What is the definition of GCD? Why are you passing a fraction instead of two numbers?	LI: Name of functions are almost always verbs and datatypes common nouns.
STUDENT: Greatest common divisor of two numbers. I have to pass the numerator and denominator. So, I'm passing the fraction. We can do it no, sir. It works.	STUDENT: Ohh, okay.
LI: Why do you think, I think, you should pass two numbers, and not the fraction.	LI: Variables are proper nouns.
1. It is nor reusable in a situation where you have to find the GCD of two numbers, and working code is not necessarily the best code.	STUDENT: Yes. But is the code right?
2. Most people will think that the GCD function will take two numbers, so code becomes less readable when you pass a fraction.	LI: What is right? Working code or well-written code?
STUDENT: Okay.	STUDENT: Well written.
	LI: Which should I check first, for working code or well-written code? In the code review, it is always STUDENT: Okay

conceptual knowledge, students were referred to internet resources (TRIN), and when the student failed to apply the conceptual knowledge, students were referred to class notes (TRCN). Students would sometimes shift the focus from the difficulty being discussed and needed to be asked to focus (TFOCUS). One student was predominantly accepting the instructor's direct answers without questioning, and there was one discussion on obeying (TOBEYVSAGREE). When other instruction methods did not result in progress, we ask the student to repeat (TREPEAT) an existing program.

Table 6. Cognitive interactions with students

Code	Description	Interaction Count
TLQ	Instructor asks leading question.	639
TDEBUG	Instructor asks the student to debug the program.	30
TEXPT	Instructor asks the student to perform an experiment.	13
TDQ	Instructor asks the student a direct question.	94
TFIB	Instructor asks the student to answer a fill in the blanks question.	13
TDA	Instructor gives a direct answer.	211
TDI	Instructor gives direct instruction.	200

8. Interactions with students in the cognitive domain

Table 6 is a summary of codes related to the cognitive domain. Table 7 shows the interaction flow from asking lead questions to direct instruction to help students construct programs. In the actual instruction, the interactions had many variations. Depending on LI's perception of the student's capability, the interaction flow happened. 518 interactions were direct instruction, and 682 interactions were attempts at constructivist approaches to learning. Many interactions resulted in students resolving their difficulties by themselves.

9. Interactions with students in the affective domain

Table 8 is a summary of interactions in the affective domain. Many of the difficulties can only be resolved by dealing with the emotions of the student. Initial emotions of fear get resolved once the student gets familiar with CS0 culture. The need for completion (TNCVSNL) interactions happens when the student submits the working code and is asked by the LI to improve the code. Students may find the request unreasonable since it delays completion and hence they need interaction to make them aware of the emotion. The interactions addressing the need for completion also happens when a student takes a lot of time to complete the program or has to park a problem

Table 7. Sample interactions to illustrate flow of interactions and coding

Message	Code
STUDENT:But how to return GCD of the numerator and denominator of a fraction?	TLQ
LI: What is the value of num and den in the function?	
STUDENT: How to assign that? Data type is int.	TLQ
LI: For which two numbers are you trying to find gcd of?	
STUDENT: Num and den.	
LI: Num and den of what?	TDQ
STUDENT: Num and den of number.	
LI: Wrong.	TDI
STUDENT: I will only tell, sir, wait.	
LI: The numerator and denominator of which fraction in the function.	TDQ
STUDENT:f1	
LI:good.	TDA

Table 8. Affective domain interaction with students

Code	Description	No. of interactions	No. of students
TCCVSCS	Instructor addressing students taking comments on code personally.	8	3
TCOUNSEL	Instructor counselling students having negative emotions	12	6
TEMOT	Instructor address student's need to know teacher emotions.	3	3
TNCVSNL	Instructor addresses need for completion vs need for learning.	13	8
TOBJECTIVITY	Instructor addresses need for objectivity.	1	1
TRELATIONSHIP	Teacher and student discuss relationship.	6	3

Table 9. Chat excerpts related to need for completion

LI: So how do you feel when you see that it is not working?	LI: Can a brain that does not complete a thought ever program?
STUDENT: Obviously, not happy. STUDENT: But what's going wrong, sir	STUDENT: Sir, you give some hint.
LI: The day such errors make you happy is when you have started loving programming. The day when you say I am going to solve is when you have started becoming a programmer.	LI: Don't you trust your brain? That if it waits, it will come up with a solution? STUDENT: Sir, I don't know.
STUDENT: Siiirrrrr.	LI: What is the hurry? STUDENT: Okay, sir.
LI: You are which branch?	LI: Nobody knows, not you, not me. When we start solving a problem, we don't know if we can ever solve it. Getting used to uncertainty is an important achievement.
STUDENT: Ise. I tried solving it, sir	STUDENT: Hmmm.
LI: Change your branch or change your attitude.	LI: Anyway, for this course, there are no marks. So we should give ourselves time to solve it.
STUDENT: But just not able to figure out. LI: But that should excite you.	STUDENT: Okay, sir
STUDENT: Okay, okay … I'll change my attitude	

for the day. Making students aware of this emotion helps them manage their time objectively. Table 9 shows the chat excerpts related to need for completion.

When students start learning in a constructivist culture, they may not make progress as fast as they wish to or as fast as their peers are progressing, and they tend to judge themselves as incapable or unsuitable for the course wrongly. The students need to be counselled that the brain needs more experiences to learn (TCOUNSEL) and that it is usual for some students to take a longer time to learn. Table 10 shows chat excerpts related to counselling.

Table 10. Chat excerpts related to counselling

STUDENT: I am really sorry, sir.	STUDENT: Sir, I am getting angry with myself. I am not getting anything. I will repeat the program again.
LI: See, it takes so much time not just for you for everybody.	LI: Listen, two things are very important for you. One, you should enjoy the learning, and two, you should accept yourself. If you are not getting it, so be it Is it because of lack of effort or sincerity?
STUDENT: Sorry, sir	
LI: Learning is a continuous process. It is not how many programs you completed. It is how many days you worked.	
STUDENT: But sir, for this, I used to spend more than 3 hrs per day.	STUDENT: Okay, sir.
LI: But you were not communicating. Communication is important, and different people are different. For some, it takes three months on one problem, all of a sudden everything becomes clear, and for some, it is clear on day one.	LI: Daily, you spend one hour on programming for the next 6 semesters. You will see the magic.
STUDENT: Fine, sir.	STUDENT: Okay, sir.

Shyness prevents interaction, and it is overcome by the demand to interact with the teacher. When there is a fault in the programs, students feel a loss of face with the teacher. There is a need to address this emotion (TCCVSCS). Consequently, the student spends time defending themselves (TDEFENSE), and making them aware of this helps them listen better. Students also need reassurance that the teacher is not angry or disappointed with the student because of the time it takes to overcome the difficulty(TEMOT). It is often important to reiterate the relationship's nature, which is non-judgmental, empathic, and respectful (TRELATIONSHIP). In one interaction, the student had to choose the right solution overcoming the respect for the teacher (TOBJECTIVITY).

10. Conclusions

The study is in a situated context and culture, and hence the time estimates and the kinds of interaction are specific to the institution. The result can be generalized to many Indian engineering institutions which are in a similar context. Though the number of students who participated in the CS0 course is small, they represent a large class of students.

The number of credits allotted to the course, the learning outcomes of the course, instruction method and teacher effort should all be aligned for successful implementation of the course. In this study, it was estimated that synchronous one on one interaction time required for teaching students to write a program for an authentic task is between 4 hours to 15 hours. When the objective is to write a program for an authentic task, the instructor will also have to address various metacognitive, affective and cognitive issues that arise during the course. This study categorizes and lists all the interactions that happened during the course. The interaction time estimates and list of kinds of interactions will help the other lab instructors plan and design their instruction.

Funding
The authors received no direct funding for this research.

Author details
Srinivasan Lakshminarayanan[1]
E-mail: srinirad@gmail.com
ORCID ID: http://orcid.org/0000-0003-2224-4156
N. J. Rao[2]
[1] Department of Computer Science and Engineering, Jain (Deemed to Be University), Jakkasandra, Ramanagara, Karnataka, India.
[2] Indian Institute of Science, Bangalore, Karnataka, India.

Citation information
Cite this article as: Types and time of interaction for teaching introductory programming using instruction method of extreme apprenticeship, Srinivasan Lakshminarayanan & N. J. Rao, *Cogent Education* (2021), 8: 1969880.

References

Ben-Ari, M. (2001). Constructivism in computer science education. *Journal of Computers in Mathematics and Science Teaching*, *20*(1), 45–73. https://www.learntechlib.org/p/8505

Campbell, J., Petersen, A., & Smith, J. (2019). Self-paced mastery learning cs1. In *Proceedings of the 50th acm technical symposium on computer science education* (pp. 955–961). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3287324.3287481

Collins, A. (1991). Cognitive apprenticeship and instructional technology. *Educational Values and Cognitive Instruction: Implications for Reform*, *1991*, 121–138.

Haatainen, S., Lakanen, A.-J., Isomöttönen, V., & Lappalainen, V. (2013). A practice for providing additional support in cs1. In *2013 learning and teaching in computing and engineering* (p. 178-183). Los Alamitos, CA, USA: IEEE Computer Society. Retrieved from https://doi.ieeecomputersociety.org/10.1109/LaTiCE.2013.39

Magrini, J. (2012). *Phenomenology for educators: Max van manen and" human science" research. Philosophy Scholarship*. https://dc.cod.edu/philosophypub/32/.

Mayer, R. E. (2004). Should there be a three-strikes rule against pure discovery learning? *American Psychologist*, *59*(1), 14. https://doi.org/10.1037/0003-066X.59.1.14

McCane, B., Ott, C., Meek, N., & Robins, A. (2017). Mastery learning in introductory programming. In *Proceedings of the nineteenth australasian computing education conference* vol 804. Springer, Cham. https://doi.org/10.1007/978-3-319-98872-6_15.

Prather, J., Pettit, R., Becker, B. A., Denny, P., Loksa, D., Peters, A., … Masci, K. (2019). First things first: Providing metacognitive scaffolding for interpreting problem prompts. In *Proceedings of the 50th acm technical symposium on computer science education* (p. 531–537). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3287324.3287374

Raj, A. G. S., Patel, J. M., Halverson, R., & Halverson, E. R. (2018). Role of live-coding in learning introductory programming. In *Proceedings of the 18th koli calling international conference on computing education research* (pp. 1–8). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3279720.3279725.

Rodríguez-Bonces, M., & Ortiz, K. (2016). Using the cognitive apprenticeship model with a chat tool to enhance online collaborative learning. *GIST Education and Learning Research Journal*, *13*(13), 166–185. https://doi.org/10.26817/16925777.318

Ureel, L. C., & Wallace, C. (2018). Board 156: Enriching communication in introductory computer science courses: A retrospective of the agile communicators project. In *2018 asee annual conference & exposition*. Salt Lake City, Utah: ASEE Conferences. Retrieved from https://peer.asee.org/29959

Vihavainen, A., Paksula, M., & Luukkainen, M. (2011). Extreme apprenticeship method in teaching programming for beginners. In *Proceedings of the 42nd acm technical symposium on computer science education* (pp. 93–98).

Wilcox, C. (2015). The role of automation in undergraduate computer science education. In *Proceedings of the 46th acm technical symposium on computer science education*. New York, NY, USA: Association for Computing Machinery. (pp. 90–95). Retrieved from https://doi.org/10.1145/2676723.2677226.

Xie, B., Loksa, D., Nelson, G. L., Davidson, M. J., Dong, D., Kwik, H., Tan, A. H., Hwa, L., Li, M., & Ko, A. J. A theory of instruction for introductory programming skills. (2019). *Computer Science Education*, *29*(2–3), 205–253. https://doi.org/10.1080/08993408.2019.1565235

Yan, L., Hu, A., & Piech, C. (2019). Pensieve: Feedback on coding process for novices. In *Proceedings of the 50th acm technical symposium on computer science education* (pp. 253–259). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3287324.3287483.

Cogent Education **(ISSN: 2331-186X) is published by Cogent OA, part of Taylor & Francis Group.**