Creating Tutorial Materials as Lecture Supplements by Integrating Drawing Tablet and Video Capturing/Sharing

Chen-Wei Wang
EECS Department, Lassonde School of Engineering, York University, Toronto, Canada
jackie@eecs.yorku.ca

ABSTRACT

We report the experience of adopting an innovative technique for creating tutorial videos which complement lectures and facilitate students' learning. Our technique relies on: **1)** preparing starter pages consisting of code fragments or writings/figures on a drawing tablet; **2)** illustrating complex ideas on the drawing tablet; **3)** recording all computer desktop activities (e.g., development of code on a programming IDE, illustration on the drawing tablet); and **4)** sharing the recorded tutorial videos with students online. Our technique has been adopted in creating tutorial series for four Computer Science and Engineering courses, ranging from the first year to the third year. Analytics of these online tutorial videos is presented to show the average amount of time which each registered student spent on watching them. Course evaluation results indicate that our technique is perceived as effective for achieving the course learning outcomes. Comparison of students' performance on complex topics (arrays and loops) also indicates a positive impact of our approach.

CCS CONCEPTS

• **Applied computing** → **Computer-assisted instruction**; • **Social and professional topics** → **Computational thinking**; **Computer science education**; **Software engineering education**.

KEYWORDS

Large Class; Laboratory Assignments; Tutorial Videos; Computational Thinking; Instructional Technologies

ACM Reference Format:
Chen-Wei Wang. 2019. Creating Tutorial Materials as Lecture Supplements by Integrating Drawing Tablet and Video Capturing/Sharing. In *Proceedings CSERC 2019 18-20 November 2019 Computer Science Education Research Conference Larnaca, Cyprus (CSERC '19), November 18–20, 2019, Larnaca, Cyprus.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3375258.3375259

1 INTRODUCTION

It is challenging to teach complex computational thinking [1, 6, 7, 28] (e.g., arrays, loops, object-oriented thinking) and software design principles (e.g., design by contract, object-oriented design patterns leveraging polymorphism and dynamic binding) in undergraduate courses, where: students have limited prior exposure to the course content; and the class size is typically large (e.g., 400+ for first-year courses, 150+ for second-year courses, and 100+ for third-year courses in our home department).

Many students encounter obstacles to full comprehension of course content because the class size restricts the instructor's intentional pauses and student interactions during lectures. To partly address this, in our recent work, we propose an innovative approach for in-class instruction of complex ideas [26], which allows the instructor to record their (verbal and written) explanations and illustrations entirely, so that students can review these materials for their learning at their own pace after class.

However, a Computer Science (CS) or Engineering course commonly has a weekly laboratory (lab), where students are assigned exercises, more challenging than examples covered in class, in order to reinforce the covered topics. However, in-class instruction is limited in two aspects, making it difficult to implement, among lectures, a logical decomposition of the taught subjects. First, lecture hours are *fixed*: very often we have to interrupt the discussion simply because the class time has run out. Second, lecture hours are *limited*: in-depth discussion and illustrations of certain technical insight cannot be accommodated in class. Consequently, there usually exists a gap between lecture materials and the pre-requisites (on concepts and skills) for completing the weekly lab assignments.

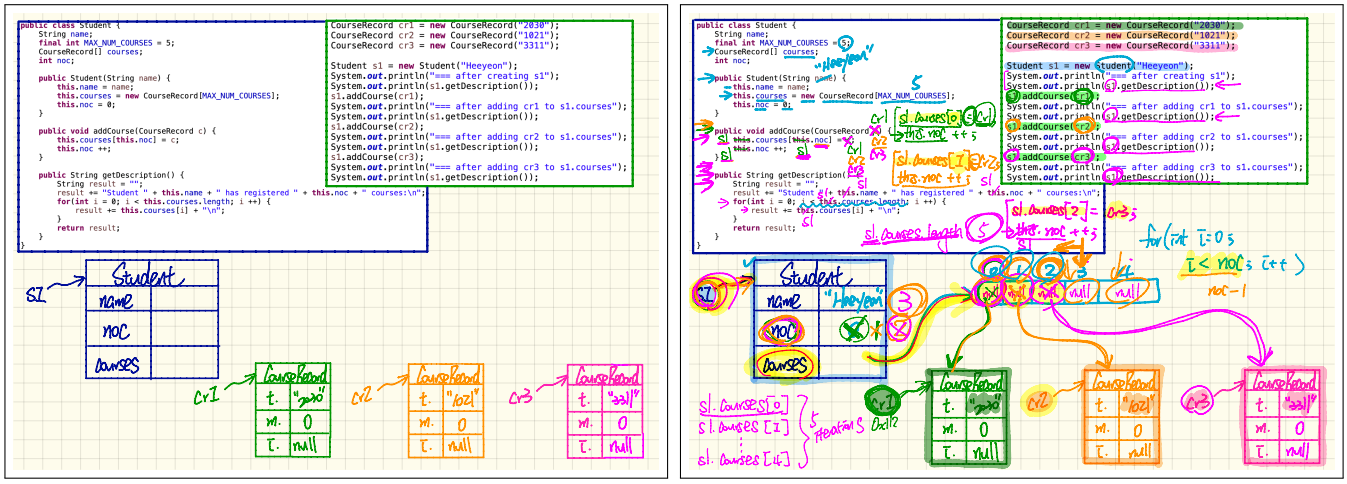
How can we make the in-depth and detailed illustrations accessible to students for their self-paced study outside the classroom, so as to help them complete the weekly lab assignments? We propose to address the two limitations of lectures by creating series of tutorial videos, each of which: **1)** is sequentialized according to the suitable logical order (as judged by the instructor); and **2)** includes in-depth remarks and illustrations on concepts and examples. For **2)**, such remarks and illustrations reflect the instructor's insight into the taught subjects, and are thus a valuable aid to student learning.

To create these tutorial videos, we have adopted, in four undergraduate CS and Engineering courses, the integrated use of: **1)** a drawing tablet for illustrating concepts and code examples; **2)** a program for recording all desktop activities, such as the slide presentation as well as illustrations on the tablet and programming IDEs; **3)** a high-end studio microphone ensuring decent sound quality of the tutorial videos; and **4)** online access to recordings and notes for students to review before attempting their lab assignments. Our tutorial videos have been uploaded to this channel: https://www.youtube.com/user/jackiechenweiwang. To complement our in-class instructions [26], we include links to these tutorial series on a lectures page: https://www.eecs.yorku.ca/~jackie/teaching/lectures/index.html.

The main contribution of this paper is a technique for creating tutorials videos on complex ideas. Our proposed technique is much more than recording the occasional annotations on a slide show.

CSERC '19, November 18–20, 2019, Larnaca, Cyprus

ACM ISBN 978-1-4503-7717-1/19/11.
https://doi.org/10.1145/3375258.3375259



(a) Before Annotations Began (26:33)

(b) After Annotations Ended (1:17:50)

Figure 1: Illustrating a Code Fragment on a Drawing Tablet (26:33 – 1:17:50 of https://youtu.be/Xuei8pOy7k8)

Instead, our technique requires the instructor to plan and prepare starter artifacts (e.g., code fragments, figures) on the drawing tablet prior to starting the recording, which is more effective than setting up these artifacts on a conventional in-class/in-office blackboard or whiteboard "on the fly".

Our proposed technique is novel in that it relies heavily on explaining and illustrating complex ideas on a drawing tablet, and that it relies on recording the process of building up complex examples from scratch. Such explanations and illustrations represent the instructor's insight into the taught subjects, as well as their thinking process which, thanks to the recording, students can review as needed and thereby learn from. As an example, Figure 1b (p2) shows an annotated fragment of object-oriented code at the end of our illustration. The reasoning process of moving from Figure 1a to Figure 1b, through recording, can be reviewed by students whenever they need.

2 TEACHING CONTEXT

We adopted our approach in four undergraduate CS and Engineering courses in the academic years of 2017 to 2019. Examples of course topics are summarized below.

CS1A Mobile Computing and *CS1B OOP: From Sensors To Actuators* are the second-semester courses for, respectively, CS and Engineering[1] students at the first year. There are 400+ students registered in each of the two courses. In both CS1A and CS1B, students learn about basic computational thinking and object orientation, but through different means. In CS1A, students develop Android mobile apps using the Android Studio IDE (Integrated Development Environment), and visualize the effects of their Java programs on physical tablets. In CS1B, students use Phidget interface boards connected to hardware equipment such an LED light bulb and Theremin glove. Example topics covered in both courses are: **1)** elementary programming (variables, data types, assignments); **2)** conditionals; **3)** loops; **4)** primitive 1D/2D arrays; and **5)** object orientation (attributes, methods, classes, and class associations).

CS2 Advanced Object Oriented Programming (with 150+ students) is the first-semester course for both CS and engineering students at the second year. Students in CS2 are required to develop, test, and debug Java programs in the Eclipse IDE. Example topics covered in CS2 are: **1)** unit testing; **2)** code reuse and subtyping via inheritance; **3)** polymorphic assignments and dynamic binding; **4)** recursion; and **5)** asymptotic upper bounds (i.e., the big-O notation) of programs.

CS3 Software Design (with 100+ students) is a required course for third-year CS and Software Engineering students. Example topics covered in CS3 are: **1)** the Design-by-Contract (DbC) method for constructing object-oriented software (using loop invariants and variants, method preconditions and postconditions, and class invariants); **2)** the information hiding design principle (exemplified by the Iterator design pattern); **3)** object-oriented design patterns leveraging polymorphism and dynamic binding (e.g., composite, visitor, observer); and **4)** introduction to program verification.

3 THE PROPOSED APPROACH

We propose an approach as visualized in Figure 2 for preparing self-paced tutorial materials (i.e., videos, illustration notes, and program source code) which facilitate both the instructor's teaching and students' learning of complex ideas (e.g., computational, design, abstract). We discuss the proposed approach from two perspectives.

First, the *instructor* chooses a set of relevant topics and creates a series of tutorial videos by presenting and illustrating those topics on their personal computer. Formats of presentation and illustration include the conventional slide show, code demonstrations on some programming IDE, as well as tracing runtime execution of program code and explaining complex logic on a drawing tablet. The entire presentation and illustration occur as various desktop activities, which are recorded, edited, and uploaded to an online sharing platform accessible to students (e.g., before attempting their weekly lab assignments). Given that illustrations on the connected drawing

[1]The majority of students in CS1B are from Computer Engineering and Software Engineering, whereas others are from Mechanical Engineering and Civil Engineering

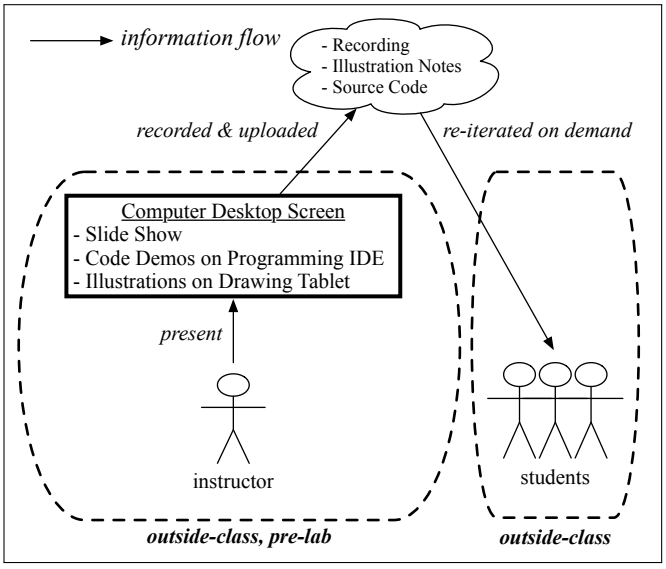


Figure 2: Creating Tutorials for Outside-Class Learning

tablet become part of the desktop activities, the instructor is able to record *thorough* discussion of pre-selected topics (e.g., a code fragment, an example to be solved from scratch) using coloured annotations. The use of a drawing tablet for illustration, although analogous to the in-class use of a whiteboard or blackboard, has the advantages of greater visibility and being "re-playable" by students.

Second, *students* follow the tablet-focused presentation on the screen of a computer or a mobile device. This means that visibility is ensured (as opposed to the case of a whiteboard or blackboard which may have limited visibility due to the size of classroom, the position of students' seats, *etc.*). Students may also download the notes from the instructor's drawing tablet (exported as PDF documents) to review the illustration process.

4 TUTORIAL SERIES

In this section, we summarize how the proposed technique was adopted to create tutorial materials for the four undergraduate courses (see Section 2 for examples of the topics and learning outcomes of the course). Twelve series of 148 tutorial videos (with a total duration of approximately 59.5 hours) have been created for the purpose of students' learning. For each tutorial series, the ordering of videos with various lengths corresponds to a logical decomposition of the taught topics.

We divide these tutorial series into two categories: **1)** study materials for lab assignments (Section 4.1); and **2)** preparation materials for lab tests (Section 4.2). All of our tutorial videos have been made available to students in this channel: https://www.youtube.com/user/jackiechenweiwang. To complement our in-class instructions [26], we include links to these tutorial series on a lectures page (where students can access slides, notes, source code, and lecture and tutorial recordings): https://www.eecs.yorku.ca/~jackie/teaching/lectures/index.html. For this paper, links to specific playlists and videos will be referenced in the following two subsections.

4.1 Study Materials for Lab Assignments

Each of the four undergraduate courses has scheduled weekly lab sessions: CS1A and CS1B have 3 hours, whereas CS2 and CS3 have 1.5 hours. Weekly lab assignments are meant for students to acquire the required practical skills (e.g., programming, object-oriented thinking, design patterns). Thus, the level of difficulty of these weekly assignments should be such that students are expected to dedicate hours before and after lab sessions to complete them.

However, given the limited number of lecture hours[2], it is challenging to fill the conceptual gap between the covered topics in class and the pre-requisites of each lab assignment. Our proposed approach attempts to solve this problem by creating self-contained tutorials on the relevant topics:

- For CS1A, students are exposed to the Android Studio programming environment on Day One of the semester to develop working apps deployable on a tablet. Assuming that students have no prior experience on programming in Java, we created the first tutorial series [17], which uses the development of a simple Body Mass Index (BMI) calculator to illustrate aspects of an event-driven controller, a graphical user interface (with simple buttons and menu boxes), and an object-oriented model. As we progress the course, three further tutorial series were created to elaborate on: **1)** separating controller and model [20]; **2)** declaring a reference-typed attribute [19]; and **3)** declaring an array whose element is reference-typed [18].
- For CS1B, students are given weekly programming assignments, expected to be completed prior to their scheduled lab sessions. Each week students are assigned four to five tutorial videos to study (each of which guiding them through the reasoning process of developing fragments of code). The lab assignments are designed in such a way that students finishing the assigned videos are able to complete the actual lab assignments independently (or with minor assistance from the teaching assistants or online forum). The tutorial series [27] contains 46 videos with the following roadmap:
  - Lab 1 (Videos 01 to 08): Simple Console Applications using Primitive Variable Assignments
  - Lab 2 (Videos 09 to 17): Simple If-Statements using the Boolean Data Type and Logical Operations
  - Lab 3 (Videos 18 to 19): A Simple Bank Account Application using Nested If-Statements
  - Lab 4 (Videos 20 to 24): Syntax and Semantics of *for*-Loops and *while*-Loops, Using Breakpoints and Debugger in the Programming IDE to Reveal Defects
  - Lab 5 (Videos 25 to 28): Basics of Arrays – Initialization using Loops and Tracing
  - Lab 6 (Videos 29 to 33): Deciding if Array Elements Universally/Existentially Satisfy Given Properties
  - Lab 7 (Videos 34 to 39): Object Orientation – Classes, Methods, Object Creations, and Method Calls
  - Lab 8 (Videos 40 to 46): Understanding and Implementing Associations between Classes

[2]There are two lecture hours per week for CS1A and CS1B, and three lecture hours per week for CS2 and CS3.

- For CS2, lab assignments require the use of classes from the Java collection library. To help students gain hands-on experience, as well as understanding the data structures of these collections (e.g., `ArrayList`, `HashTable`), we created a tutorial series [21] for them to review before attempting the lab assignments.
- For CS3, the composite/visitor design patterns are expected to be used in one of the labs and projects. Due to the limited number of lecture hours, we cannot guide students through the process of implementing these two advanced design patterns. Instead, we created a tutorial series [16] which implements and debugs a simple language processor using the two design patterns. For some labs and the project, we adopt a programming framework [10] which restricts all students to work under a given API, while being allowed to design their own programming modules that implement the common API. This programming framework is challenging for students to use due to its sophisticated architecture. To help students get started, we created a tutorial series [24] which guides them through the use of the framework: architecture, extension, regression testing, and debugging.

In all CS1B, CS2, and CS3, we require students to apply the common software engineering practice of managing their projects using a revision control system such as Github. We created a tutorial series [22] to help them initiate a private account and workspace on their computers, as well as understand the workflow of common operations (e.g., `clone`, `commit`, `push`, `pull`).

4.2 Preparation Materials for Lab Tests

An important learning outcome of CS1A, CS1B, and CS2 is being able to write *runnable* programs (upon which students are assessed through automated unit tests). We emphasize to students that when they write an essay, if there are grammatical mistakes, it can still be interpreted by a human. Computer programs, on the other hand, just cannot be run (and hence undefined runtime behaviour) when they contain compile-time syntax or type errors.

In order to help students (especially those in CS1A and CS1B who have little prior programming experience and discipline) write compilable code during in-lab computer tests, we created a tutorial series [25] to guide them through the process. For CS1A and CS1B, we show how to write valid Java methods, given: **1)** an API **2)** a console application tester; and **3)** expected console outputs. For CS2, we show how to write valid classes/methods, given a set of unit tests.

Furthermore, in order to help students apply the above code-writing process to solve real problems:

- For CS1A and CS1B, we created a tutorial series [23] on going through the code and thinking process for solving twelve practice problems (involving arrays and loops).
- For CS2, we created a tutorial series [15] on developing a complete Birthday Book application (using two parallel arrays with methods for insertions, removals, and lookups).

5 A PATTERN FOR TUTORIALS

The majority of our tutorial videos (Section 4) conform to the following general pattern:

(1) **Present the Problem.** This can be done by using a slide show to present the general problem to be solved, by using a programming IDE to demonstrate what is ultimately expected from the final (e.g,. software) product, or even by just pointing to what has been achieved in the previous tutorial video(s).
(2) **Sketch the Solution.** This part emphasizes the high-level thinking process, which can be illustrated on the drawing tablet, which may be pre-set with starter pages containing, e.g., code fragments, formulas, writings, figures.
(3) **Develop the Solution.** This is typically done in a programming IDE, or any software tool that is applicable to the course being taught.
(4) **Discuss the Solution.** This is to be done on starter pages on the drawing tablet. These starter pages may be set up either before the recording starts, or after Step 3[3] (by copying and pasting snapshots of parts of the solution developed). As the discussion progresses, we annotate on the starter pages to gradually build towards the solutions or conclusions.

The above pattern requires the instructor to determine what concepts/examples they will illustrate in the same series of tutorial videos, and then to create the starter pages on the drawing tablet accordingly. Some steps (except tablet illustrations) may be omitted, and the order of choreographing these components may be adjusted according to the subject being taught.

As an example of instantiating the above pattern, consider Video 42 from the Java Tutorial Series for CS1B [27]: https://youtu.be/Xuei8pOy7k8. This tutorial video shows how to implement `Student` objects, each of which stores an array of `CourseRecord` objects:

- 00:00 – 03:03: Summarize classes and methods developed in the previous videos, and briefly mention the extension to be completed in the current tutorial video.
- 03:04 – 26:32: Develop the programming solution on Eclipse right away (and sketch the idea later).
- 26:33 – 47:05: On the drawing tablet, trace the developed code line by line, by visualizing object creations and method calls. This part was actually recorded separately and appended to the previous recording, so that it was possible to take snapshots of the code developed between 03:04 and 26:32, and to paste them to starter pages on the tablet.
- 47:06 – 50:49: On Eclipse, extend the code by introducing a second version of the implemented methods.
- 50:50 – 58:04: On the drawing tablet, sketch the idea about the second version of implementation.
- 58:05 – 59:57: On Eclipse, execute the second version of implementation, faulty due to a null pointer.
- 59:58 – 1:07:00: On the drawing tablet, illustrate how the runtime exception occurs and go back to Eclipse to fix the code accordingly.
- 1:07:01 – end: On the drawing tablet, justify why the final implementation works in two boundary cases: empty array vs. fully-occupied array.

[3] This alternative would require editing of the recordings.

Contrast Figure 1a with Figure 1b on page 2 to see how much illustrations of complex ideas has been performed in the above tutorial video. There are many more instantiations of the above pattern that can be found from our series of tutorials videos (Section 4).

6 ADOPTING THE APPROACH

Figure 3 summarizes how to assemble the equipment to implement the proposed approach. Here we describe what we use, but the interested reader may choose other equipment with the same functionality.

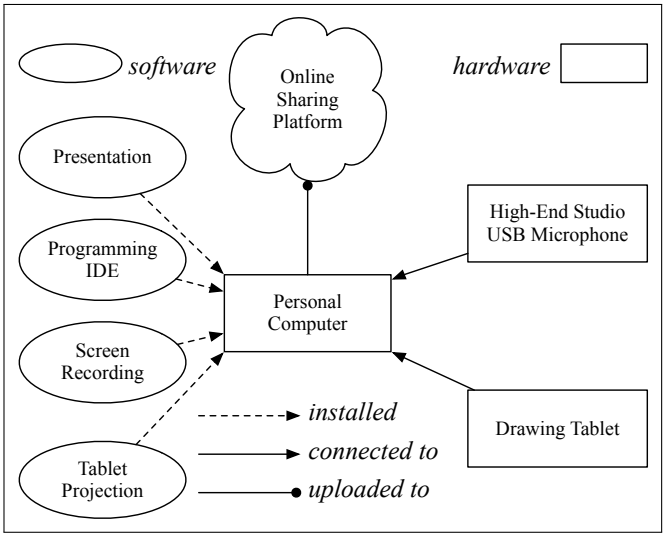


Figure 3: Adopting the Approach: Schematic View

Install the following software programs on your teaching computer (e.g., a MacBook): **1)** a presentation program (e.g., a PDF or PowerPoint reader) for your slides; **2)** a programming IDE as applicable to your course (e.g., Android Studio, Eclipse); **3)** a screen recording program (e.g., Active Presenter [11] for recording all desktop activities on the computer; and **4)** a program for projecting the screen of your drawing tablet (e.g., the free QuickTime Player).

Connect the following hardware to your computer: **1)** a high-end studio microphone (e.g., Blue Yeti [2]) using a USB cable; and **2)** a drawing tablet (e.g., iPad Pro) installed with an app for annotations (e.g., GoodNotes, Notability). For **2)**, a wired connection to the USB port is recommended for stability throughout the recording session. To project the screen of the drawing tablet to your computer desktop, if you use the QuickTime player and an iPad Pro, start a "New Movie Recording" and select your iPad as the camera.

When ready to start your tutorial recording, start the screen recording program and choose the connected microphone as the input device. Our experience has shown that a small amount of *basic* editing is necessary in order to: **1)** add a cover page (with the instructor's information, topics, *etc.*); and **2)** combine parts of recordings into an integral unit. As we become more experienced in preparing the starter artifacts and recording long sessions, item **2)** becomes less common as we need not stop because of errors or imperfections on the explanations or illustrations.

When each recording session is finished, stop the screen recording, export it to an acceptable form (e.g., MP4), upload it to an online video sharing platform (e.g., YouTube), add it to the relevant playlist, and publish the link to students. The annotation app on your drawing tablet should allow you to export the annotated notes (e.g., Figure 1b, p2) as a PDF file.

7 EVALUATIONS

7.1 Student Engagement

Table 1 summarizes on YouTube, as of April 2019 when all courses were completed: **1)** the average number of minutes which each registered student spent watching the videos[4]; and **2)** the average completion rate (i.e., the ratio of average watch time to the duration of the tutorial series in question) accordingly. The average time is calculated based on one iteration of CS1A (Winter 2018 with 357 students[5]), one iteration of CS1B (Winter 2019 with 459 students), two iterations of CS2 (Fall 2017 with 99 students and Fall 2018 with 134 students), and three iterations of CS3 (Fall 2017 with 82 students, Fall 2018 with 88 students, and Winter 2019 with 95 students).

Course	Series	Avg. Watch Time (Min)	Completion Rate
CS1A	[17]	304.58	86.52%
	[20]	42.48	45.34%
	[19]	31.25	65.04%
	[18]	75.83	18.33%
CS1B	[27]	365.36	21.15%
CS1A,B	[23]	35.33	14.23%
CS2	[15]	108.67	41.21%
	[21]	28.00	34.50%
CS3	[24]	58.9	48.59%
	[16]	25.1	22.80%
CS1B,2,3	[22]	35.06	43.62%

Table 1: Average Watch Time and Completion Rates

In Table 1, the measures of average watch times, and of completion rates accordingly, are arguably underestimates: apathetic students (e.g., those who never watched any of the videos) are not excluded. Consequently, a "good" student (e.g., those who attempted to watch these videos) in these courses should have a higher completion rate. Such engagement is confirmed by many students in the (informal) midterm and (formal) end-of-semester course evaluations, expressing that these tutorial videos are helpful.

7.2 Improvement on Performance

Our proposed approach to making tutorial videos is meant for helping students understand complex computational thinking. One example is writing procedural code (in Java) using primitive arrays and loops. Table 2 shows the results of two in-lab computer tests in Winter 2018[6] from CS1A (taught by us, and our tutorial series on practice test solution [23] was supplied prior to the lab test) and CS1B (not taught by us, and no tutorial videos were supplied).

[4]We exclude [25], which is not directly related to computational thinking.
[5]There was an abnormal drop on the number of students due to a labour disruption.
[6]The lab test for CS1B had more participants because it was taken prior to a labour disruption, whereas the lab test for CS1A was taken during the remediation period.

Course	Tutorials?	# of Students	Avg. Performance
CS1A	Yes	201	57.7%
CS1B	No	439	43.75%

Table 2: Performance Comparison: Arrays and Loops

As can be observed from Table 2, our tutorial solutions [23] had a positive impact on CS1A students. Although the two tests in Table 2 used different questions, the level of difficulty of tasks in CS1A (e.g., given as inputs two sorted arrays, return a new sorted array that merges them) is significantly higher than that of tasks in CS1B (e.g., given as inputs a list of numbers and an integer n, return a sublist whose values are larger than or equal to n).

7.3 Student Feedback

The anonymized online course evaluations[7] of CS1A, CS1B, CS2, and CS3 indicate that our tutorial videos (Section 4 and Section 5), despite their length, are perceived by students as effective:

- "*... coming into the course knowing nothing about java, his online tutorials allowed me to understand the course material ... . It was easy to follow, very in-depth with the explanations and most importantly, had relevance to the lab and course syllabus.*" [CS1B]
- "*He [the author] puts in a lot of effort to get the students involved in the course material and also makes very long tutorial videos for us to really understand and concepts.*" [CS1B]
- "*The best things about this course are] Just the way he explains everything in the tutorial videos. Tracing code line by line makes it so helpful and easy to understand[.]*" [CS1B]
- "*The lecture recordings were immensely helpful and the tutorial videos for the most part were crucial to my success ...*" [CS1B]
- "*... the tutorial videos. They really helped me understand the skills necessary to learn the course objectives.*" [CS1B]
- *"The tutorial videos explained alot of concepts."* [CS1B]
- *"The prof is the best, lots of resources provided. Made difficult concepts easier to understand through all methods, tutorial video, iPad illustrations, test cases, etc"* [CS1B]
- *"Jackie is a very passionate and respected professor. He puts in a lot of effort to get the students involved ... and also makes very long tutorial videos for us to really understand the concepts."* [CS1B]
- "*Jackie prepares online tutorial to help students. And he recorded his lectures, so that students can save time coming to lecture. Some say those videos are too long. But to me, they are essential and neccesary. I really appreciate those videos and efforts Jackie put in making them. Thanks professor.*" [CS1B]
- *"[The best part of the course is] Just the way he explains everything in the tutorial videos. Tracing code line by line makes it so helpful and easy to understand Probably the best coding professor so far."* [CS1B]
- *"The course was hard hut the tutorial video were too helpful[.]"* [CS1B]
- *"The instructor was very organized and helpful, he would post all lecture materials and record java tutorials out of his own time to support the students in the lab exercises. He did a great job in teaching the course for beginner programmers[.]"* [CS1B]
- *"He really puts in the effort and wants us to succeed which is simply fantastic. I have never seen a professor put that much time into making tutorial videos for us. (Although some of us just can't get it in our heads) Thanks Jackie!"* [CS1B]
- *"The professor was very great. He was very knowledgable and had a passion for the course which greatly encouraged others to take part in the course. He would also take much time out of his own schedule to post weekly tutorial videos which greatly covered every little detail of every concept."* [CS1B]
- *"Professor Wang is great! The tutorial videos are an immense help and I actually understand what is going on in the course."* [CS1B]
- "*This course is easy to understand. Also, tutorial videos are better than the lectures.*" [CS1A]
- "*[The best things about this course are] Professor and his teaching style! he is excellent! the thing that he records every lecture and tutorials enable students to go over the material as much as they need.*" [CS1A]
- "*... video tutorials are extremely useful (although very long and time consuming to watch), and he took the time to explain concepts thoroughly and in detail which was helpful to complete labs and further my understanding.*" [CS1A]
- "*The tutorial videos were also great because he led us step by step of the way of a very new and complicated android application development process.*" [CS1A]
- "*The instructor did his best for understanding the course materials. Specially, all of his tutorial videos were very helpful to me to fulfill learning outcomes. I highly appreciate the instructors efforts toward the student.*" [CS1A]
- "*The extra tutorial videos helped a lot in understanding some of the concepts and how to complete the labs.*" [CS1A]
- "*The extensive online tutorials for every new topic in the course was very helpful.*" [CS1A]
- "*Great teacher, really good at explaining difficult concepts. Made the course understandable for everyone. Labs were designed well to work in unison with the youtube tutorial videos.*" [CS1A]
- "Very excellent teaching style, the youtube tutorials and available notes made for a great experience." [CS2]
- "*The tutorial series and the recording system help me a lot in this course.*" [CS2]
- "*[The best things about this course are] The volume of material provided by the professor. Between the lecture notes, the recorded lectures, and tutorial videos there was more than enough material to ensure I could learn about the required topics.*" [CS2]
- "*Professor Jackie is a great instructor. The recordings of his lectures and extra tutorials are helpful and he is always very nice and courteous with his students.*" [CS2]
- "*I admire how passionate the instructor when he is teaching the course. Also, how dedicated the instructor in helping students to do well in this hard course by accommodating additional review sessions and online video tutorials.*" [CS2]
- "*... really great to have the tutorial videos for the labs, it really helped us to where we needed to start for the lab.*" [CS3]
- "*The professor did a great job at explaining the course material as well as provided vast amount of resources for the students to succeed in class such as videos and tutorials plus a lot of office hours.*" [CS3]
- "*The professor was very helpful even though the language and environment used were not great. He made videos for us to learn from and tutorials to follow along ...*" [CS3]
- "*[The best things about this course are] The availability of video lectures and tutorials.*" [CS3]
- "*It was also really great to have the tutorial videos for the labs, it really helped us to where we needed to start for the lab.*" [CS3]
- "*Jackie explains it [the Eifel design language] well enough and he has enough tutorials and resources on his website that it is possible to learn the language over the span of the course.*" [CS3]
- "*The professor put lots of effort into making tutorial video to help us faster adapter the Eiffel language.*" [CS3]

[7] We do not include numerical ratings because it is hard to distinguish between the impact of our in-class instruction and that of our tutorial series.

8 REFLECTION

Drawing Tablet vs. Blackboard/Whiteboard. By using a drawing tablet, we can teach more effectively by creating starter pages, and then gradually adding annotations as illustrated between Figure 1a and Figure 1b. This is essentially a *dynamic* way for controlling the pace and level of details that the instructor judges appropriate for how the current class is understanding the materials. Furthermore, the ability to draw and annotate with the various colours (e.g., red-underline a line of code, colour a portion of drawing) helps the instructor communicate key points to students.

Alternatively, many excellent instructors prefer to creating their tutorial videos by recording them standing in front of a blackboard/whiteboard, which may also be pre-set with writings or drawings. However, when the current discussion is finished and before a new discussion can be started, the recording must be interrupted in order to clear and setup the blackboard/whiteboard. On the other hand, our use of an annotation app allows us to preset multiple starter pages on the drawing tablet, and to launch them without interruptions in a single recording session. Moreover, starter pages of a drawing tablet in our case are *digital* and can thus be reused and elaborated over time.

Instructor's Required Efforts. Our proposed approach relies heavily on the instructor's determination on dedicating time and efforts to: **1)** planing what concepts/examples they will illustrate in each tutorial video in the same series; **2)** creating all necessary starter pages on the drawing tablet accordingly; and **3)** choreographing a logical interleaving between various explanations and/or illustrations (e.g., on a programming IDE, on the drawing tablet, on a slide show, *etc.*). Rather than simply writing/drawing illustrations/explnations "on they fly", we advise that the instructor spend time carefully planning the layout of each starter page. Most valuably, these starter pages, once created, may be *elaborated* over time and later *reused* for teaching the same or similar subjects.

Lack of Student Interaction. Lectures do not scale well for student learning because: **1)** in a large venue there is a negative effect on visibility and audibility of the presentation; **2)** a large class restricts interaction between students and the instructor; and **3)** in-depth, comprehensive explanations and illustrations cannot be fit into the limited time. Our recent work [26] partly addresses **1)** and **3)**. Our proposed approach for creating tutorial videos in this paper intends to address **1)** and **3)** further. Addressing **2)**, for large classes, is beyond the scope of this paper.

9 RELATED WORKS

We report a novel technique for creating tutorial videos which complement lectures and facilitate students' learning. Some recommendations for creating engaging tutorial videos from [4] — "continuous visual flow" and "the instructor speaks ... with high enthusiasm" — correspond to the guiding principles of creating our tutorial series. However, also as indicated in [4], videos shorter than six minutes are more effective for students' engagement. Some students in CS1B (close to 20% of those who completed the online evaluation) complained that the lengthy videos increased their course workload. Nonetheless, it was only a much smaller group of students (less than 5%) expressing that these videos are unnecessary or useless for them to achieve the course learning outcomes.

Our tutorial videos offer new, more sophisticated examples, which cannot be completed in class, rather than repeating demonstrations done in class [12]. Consequently, given that our tutorial videos are meant for thoroughly demonstrating sophisticated examples, they are not comparable to short "podcast highlights" or full-length lecture footage [8]. Moreover, unlike many other attempts of making better tutorial videos [5, 9], our approach is meant for teaching complex computational thinking [1, 6, 7, 28] by requiring the careful setup of starter artifacts (e.g., code fragments, figures, writings) on a drawing tablet for illustrations, which are recorded for students to review outside class. Our videos were perceived as effective (see Section 7.3 for examples).

Stanford Online [14], designed for online distance learning, offers similar tutorial support. The use of a drawing tablets for intensive illustrations, as well as constant switches between the various desktop activities, is not typical for courses there. Similar to Stanford Online are the online course repositories such as Coursera [3] and Udemy [13], but these are meant to be commercial, unlike our intention. Moreover, the use of intensive tablet illustrations is also not typical in these commercial courses.

10 CONCLUSION

In this paper, we report the experience of adopting an innovative technique for creating tutorial videos on complex ideas which complement lectures and facilitate students' learning. Our proposed technique is novel in that it relies heavily on explaining and illustrating complex ideas on a drawing tablet, and that it relies on recording the process of building up complex examples from scratch. Such explanations and illustrations represent the instructor's insight into the taught subjects, as well as their thinking process which, thanks to the recording, students can review as needed and thereby learn from. Furthermore, our proposed technique is much more than recording the occasional annotations on a slide show. Instead, our technique requires the instructor to plan and prepare starter artifacts (e.g., code fragments, figures) on the drawing tablet prior to starting the recording, which is more effective than setting up these artifacts on a conventional in-class/in-office blackboard or whiteboard "on the fly".

Our technique has been adopted in creating tutorial series for four Computer Science and Engineering courses, ranging from the first year to the third year. Analytics of these online tutorial videos is presented to show the average amount of time which each registered student spent on watching them. Course evaluation results indicate that our technique is perceived as effective for achieving the course learning outcomes. Comparison of students' performance on complex topics (arrays and loops) also indicates a positive impact of our approach.

As future work, we will: **1)** distribute a questionnaire specific to the learning experience of our tutorial videos; **2)** conduct more performance comparison on other subjects; and **3)** reflect on the proposed pattern (Section 5) by creating tutorials for other CS or Engineering courses (e.g., a course on the introduction to theory of computation). Furthermore, we may also investigate why or why not our development of long, thorough tutorial videos helps the learning process of students in under-represented groups (who maybe perform relatively poorly in large classes).

REFERENCES

[1] Jens Bennedsen, Michael E. Caspersen, and Michael Klling. 2008. *Reflections on the Teaching of Programming: Methods and Implementations* (1 ed.). Springer Publishing Company, Incorporated.

[2] Blue. [n.d.]. Blue Yeti: Professional Multi-Pattern USB Mic for Recording and Streaming. https://www.bluedesigns.com/products/yeti/.

[3] Coursera. [n.d.]. https://www.coursera.org/.

[4] Philip J. Guo, Juho Kim, and Rob Rubin. 2014. How Video Production Affects Student Engagement: An Empirical Study of MOOC Videos. In *Proceedings of the First ACM Conference on Learning @ Scale Conference* (Atlanta, Georgia, USA) *(L@S '14)*. ACM, New York, NY, USA, 41–50. https://doi.org/10.1145/2556325.2566239

[5] Juho Kim. 2013. Toolscape: Enhancing the Learning Experience of How-to Videos. In *CHI '13 Extended Abstracts on Human Factors in Computing Systems* (Paris, France) *(CHI EA '13)*. ACM, 2707–2712. https://doi.org/10.1145/2468356.2479497

[6] Anna Lamprou and Alexander Repenning. 2018. Teaching How to Teach Computational Thinking. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education* (Larnaca, Cyprus) *(ITiCSE 2018)*. ACM, New York, NY, USA, 69–74. https://doi.org/10.1145/3197091.3197120

[7] James Lockwood and Aidan Mooney. 2017. Computational Thinking in Education: Where does it Fit? A systematic literary review. *CoRR* abs/1703.07659 (2017). arXiv:1703.07659 http://arxiv.org/abs/1703.07659

[8] Mia Minnes, Christine Alvarado, Max Geislinger, and Joyce Fang. 2019. Podcast Highlights: Targeted Educational Videos From Repurposed Lecture-capture Footage. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. ACM, 365–371. https://doi.org/10.1145/3287324.3287465

[9] Cuong Nguyen and Feng Liu. 2015. Making Software Tutorial Video Responsive. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (Seoul, Republic of Korea) *(CHI '15)*. ACM, 1565–1568. https://doi.org/10.1145/2702123.2702209

[10] J. S. Ostroff and C. Wang. 2018. Modelling and Testing Requirements via Executable Abstract State Machines. In *2018 IEEE 8th International Model-Driven Requirements Engineering Workshop (MoDRE)*. 1–10. https://doi.org/10.1109/MoDRE.2018.00007

[11] Active Presenter. Version 7. All-in-one Screen Recorder, Video Editor & eLearning Authoring Software. https://atomisystems.com/activepresenter/

[12] Ben Stephenson. 2019. Coding Demonstration Videos for CS1. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. ACM, 105–111. https://doi.org/10.1145/3287324.3287445

[13] Udemy. [n.d.]. https://www.udemy.com/.

[14] Stanford University. [n.d.]. Stanford Online. https://online.stanford.edu/courses.

[15] Chen-Wei Wang. 2017. Developing a Birthday Book Application in Java: Console Tester vs. JUnit Tests. https://www.youtube.com/playlist?list=PL5dxAmCmjv_6USwWU8e9XrJYZNShNSEmw.

[16] Chen-Wei Wang. 2017. Implementing the Composite and Visitor Design Patterns. https://www.youtube.com/playlist?list=PL5dxAmCmjv_4z5eXGW-ZBgsS2WZTyBHY2.

[17] Chen-Wei Wang. 2018. Developing a BMI Calculator: Model, View, Controller. https://www.youtube.com/playlist?list=PL5dxAmCmjv_7WvY_QnJrcPczM_KjABxBn.

[18] Chen-Wei Wang. 2018. Developing a Model Using Array-Typed Attributes. https://www.youtube.com/playlist?list=PL5dxAmCmjv_6KPR7g2mcUh7OwQML7RG1M.

[19] Chen-Wei Wang. 2018. Developing a Model Using Reference-Typed Attributes. https://www.youtube.com/playlist?list=PL5dxAmCmjv_6I1AoUbxkwUd5OiDZFA6IJ.

[20] Chen-Wei Wang. 2018. Even-Driven Controller vs. Object-Oriented Model. https://www.youtube.com/playlist?list=PL5dxAmCmjv_4qD4f9le4HMsji7ltReldu.

[21] Chen-Wei Wang. 2018. Java Collection Library. https://www.youtube.com/playlist?list=PL5dxAmCmjv_4rOxjfTfIxNp42vO8SnT8n.

[22] Chen-Wei Wang. 2018. Managing Software Projects Using Github. https://www.youtube.com/playlist?list=PL5dxAmCmjv_58KxTSd1CRbpinmSF8EPJx.

[23] Chen-Wei Wang. 2018. Solutions to Practice Test on Arrays and Loops. https://www.youtube.com/playlist?list=PL5dxAmCmjv_4UZNiLzeFPAgDDv2vLCGb4.

[24] Chen-Wei Wang. 2018. Use of the Eiffel Testing Framework (ETF). https://www.youtube.com/playlist?list=PL5dxAmCmjv_5unIgLB9XiLwBey105y3kI.

[25] Chen-Wei Wang. 2018. Writing Valid Java Code Based on Given Tests. https://www.youtube.com/playlist?list=PL5dxAmCmjv_66n9HGJ3tlzPbX3KYt_8zs.

[26] Chen-Wei Wang. 2019. Integrating Drawing Tablet and Video Capturing/Sharing to Facilitate Student Learning. In *Proceedings of the ACM Conference on Global Computing Education* (Chengdu,Sichuan, China) *(CompEd '19)*. ACM, New York, NY, USA, 150–156. https://doi.org/10.1145/3300115.3309530

[27] Chen-Wei Wang. 2019. Java: from Procedural to Object-Oriented Programming. https://www.youtube.com/playlist?list=PL5dxAmCmjv_5NRNPG3OiWZWAqmvCjiLfG.

[28] Jeannette M. Wing. 2006. Computational thinking. *Commun. ACM* 49, 3 (2006), 33–35. https://doi.org/10.1145/1118178.1118215