Integrating Drawing Tablet and Video Capturing/Sharing to Facilitate Student Learning

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 in-class instruction. The technique relies on: **1)** replacing the blackboard/whiteboard by a portable drawing tablet; **2)** preparing starter pages consisting of code fragments or writings/figures on the drawing tablet for in-class illustrations on complex ideas; **3)** recording the in-class illustrations on the drawing tablet for students to review the thinking process after class. This technique has been adopted in three Computer Science and Software Engineering courses, ranging from freshman to junior years, and the student evaluation results indicate that this technique is effective and helps students achieve the course learning outcomes. Comparison of student performance on complex ideas also indicates a positive impact of our approach.

KEYWORDS

Communication Skills; Computational Thinking; Instructional Technologies; Learning Environment; Undergraduate Instruction

ACM Reference Format:
Chen-Wei Wang. 2019. Integrating Drawing Tablet and Video Capturing/Sharing to Facilitate Student Learning. In *ACM Global Computing Education Conference 2019 (CompEd '19), May 17–19, 2019, Chengdu,Sichuan, China.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3300115.3309530

1 INTRODUCTION

It is challenging to teach complex computational thinking [2, 5, 6, 14] (e.g., nested loops on 2D arrays, recursion) and design principles (e.g., design by contract, object-oriented design patterns leveraging polymorphism and dynamic binding) in undergraduate courses, when the students have limited prior exposure to the course content. The class size of these courses is typically large (e.g., 400+ for freshman courses, 150+ for sophomore courses, and 100+ for junior courses in our department). Many students encounter obstacles to full comprehension of course content because: **1)** the class size restricts the instructor's intentional pauses and student interactions; and **2)** students are occupied by copying (often blindly) the instructor's remarks and board notes. For **2)**, such remarks and notes reflect the instructor's insights into the taught subjects, and are thus a valuable aid for student learning.

How can we make the in-depth and detailed illustrations in class accessible to students for their self-paced study outside the classroom? To address this question, we support student learning by allowing them, outside the classroom, to review contents taught in class (presented verbally and in written form). To achieve this, we have adopted, in three undergraduate Computer Science (CS) courses, the integrated use of: **1)** a drawing tablet (replacing the blackboard/whiteboard) 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 wireless microphone allowing the instructor to move around the classroom without compromising the recorded sound quality; and **4)** online access to recordings and notes for students to review the class.

Our proposed teaching technique is novel in that it replaces a conventional blackboard/whiteboard with a *portable* drawing tablet, and it relies on recording the process of building up complex examples (e.g., static software architecture, dynamic runtime execution) from scratch. Such illustrations and examples represent the insight into the taught subjects and thinking process which, due to the recording, students can review as needed and thereby learn from. We maintain a website for students to access the recordings and illustration notes after class, and even after completing the courses: https://www.eecs.yorku.ca/~jackie/teaching/lectures/index.html. As an example, Figure 2b (p4) shows an annotated fragment of code at the end of our illustration on a 2D array. The reasoning process of moving from Figure 2a to Figure 2b, through recording, can be reviewed by students whenever they need.

Nonetheless, our proposed teaching technique cannot be implemented via a smart board (e.g., [11]). First, the installation of a smart board, similar to that of a conventional blackboard/whiteboard, has limited visibility due to the size of classroom and the position of students' seats. Second, it is not effective for the instructor to build up illustrations on complex examples, requiring a large amount of hand writing, on the touch screen of a smart board. Third, it is not yet the standard practice for a classroom to be equipped with a smart board: our technique with a portable drawing tablet can be adopted in any classroom with a standard projector.

The main contribution of this paper is a technique (as exemplified in Section 4) for setting up in-class illustrations on complex ideas. Our proposed technique is much more than the occasional and lightweight annotations on an in-class slide show using a tablet computer. Instead, our technique requires the instructor to prepare starter artifacts (e.g., code fragments, figures, an enumeration of theorems) on the drawing tablet prior to each class, so as to save time on setting up these artifacts on a conventional in-class blackboard/whiteboard "on the fly". Such carefully prepared starter artifacts provide the basis for building up the illustrations in class.

CompEd '19, May 17–19, 2019, Chengdu,Sichuan, China

ACM ISBN 978-1-4503-6259-7/19/05…$15.00
https://doi.org/10.1145/3300115.3309530

The rest of the paper is organized as follows. Section 2 summarizes the topics and learning outcomes of the three undergraduate CS courses, in which we adopted the proposed approach. Section 3 discusses our proposed approach. Section 4 describes an example of adopting our approach. Section 5 presents results of course evaluations and performance comparison. Section 6 outlines the equipment requirements. Section 7 reflects on our experience. Section 8 discussed the related works. Section 9 concludes the paper.

2 TEACHING CONTEXT

The proposed approach is meant for effective teaching (for instructors) and learning (for students) of courses involving complex computational thinking or abstract theories. For example, in the academic year of 2017 – 2018, we adopted this approach in three undergraduate, computer science courses. Examples of some of the course topics and course learning outcomes (CLOs) are summarized below.

2.1 First-Year Course: Introductory OOP

CS1 Mobile Computing (with 400+ students) is the second-semester course for CS[1] students at the first year. Students learn about basic computational thinking and object orientation through developing Android mobile apps using the Android Studio IDE (Integrated Development Environment), and visualizing the effects of their Java programs on physical tablets. Example topics covered in CS1 are: **1)** elementary programming (variables, data types, assignments); **2)** conditionals; **3)** loops; **4)** primitive 1D and 2D arrays; and **5)** object orientation (attributes, methods, classes, and class associations).

Some CLOs of CS1 are: **CLO1)** Understand software development within an object-oriented framework using a modern programming language and tool set; and **CLO2)** Use a set of computing skills such as reasoning about algorithms, tracing programs, test-driven development, and diagnosing faults.

2.2 Second-Year Course: More Advanced OOP

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 their Java programs in the Eclipse IDE. Example topics covered in CS2 are: **1)** unit testing (using JUnit); **2)** code reuse and subtyping via inheritance; **3)** polymorphic assignments and dynamic binding; **4)** recursion; **5)** asymptotic upper bounds (i.e., the big-o notation) of programs; and **6)** implementations of simple data structures such as singly-linked lists, stacks and queues, and binary trees.

Some CLOs of CS2 are: **CLO1)** Implement aggregations and compositions; **CLO2)** Implement inheritance; **CLO3)** Use recursion; and **CLO4)** Implement linked lists.

2.3 Third-Year Course: Software Design

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 [7] (using loop invariants and variants, method preconditions and postconditions, and class invariants); **2)** the information hiding design principle [9] (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 using Hoare Triples [4].

Some CLOs of CS3 are: **CLO1)** Describe software specifications via Design by Contract; **CLO2)** Implement specifications with designs that are correct, efficient and maintainable; and **CLO3)** Design software using appropriate abstractions, modularity, information hiding, and design patterns.

[1]The corresponding Java course for engineering students uses fidget boards connected with hardware equipment such as LED light bulbs.

3 THE PROPOSED APPROACH

This paper proposes an approach visualized in Figure 1 to both effective teaching (for instructors) and learning (for students) of complex ideas (e.g., computational, design, abstract). We discuss the proposed approach from two perspectives.

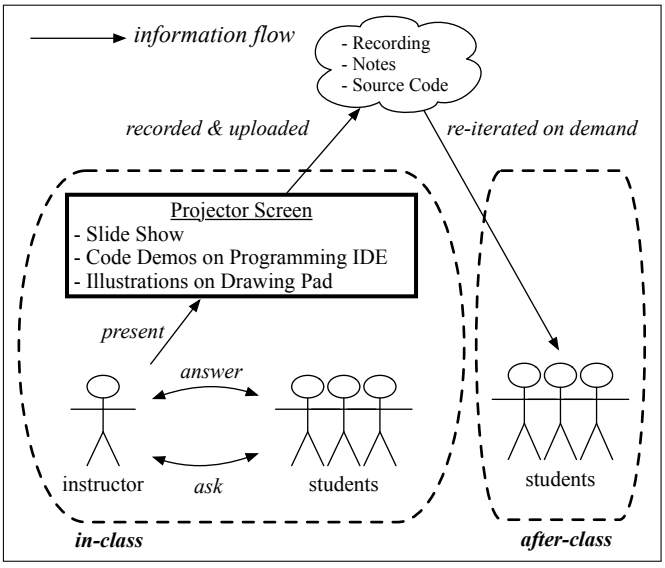


Figure 1: Proposed Approach: Effective After-Class Learning

The *instructor* presents course materials on a personal computer. The entire presentation, as desktop activities of the instructor's computer, is not only projected to the projector screen for students to follow in class, but also recorded (and later uploaded and made accessible) for students to review after class. Formats of presentation include the conventional slide show and code demonstrations on some programming IDE. Furthermore, a drawing tablet connected to the instructor's computer replaces the conventional whiteboard or blackboard (See Section 6 for its configuration). Given that illustrations on the connected drawing tablet become part of the desktop activities, the instructor is able to record: **1)** discussion of pre-selected topics (e.g., a code fragment, an example to be solved from scratch) using coloured annotations; and **2)** answers to students' questions that require detailed illustrations.

The *students* follow the tablet-based presentation via the same projector screen as the slideshow meaning 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). With the understanding that the entire presentation is recorded and will later be made accessible for review at their own pace, students may focus more on thinking through the contents and asking questions accordingly, without being distracted by copying (often blindly) the instructor's written or verbal notes.

4 EXAMPLE: TEACHING 2D ARRAYS

In this section, we illustrate how the proposed technique was adopted to teach the topic of two-dimensional arrays in CS1 (see Section 2 for examples of the topics and learning outcomes of the course). The same approach was also adopted for teaching CS2 and CS3.

Consider the following problem: Given a two-dimensional table specifying the distances between cities (where rows and columns denote, respectively, departure and destination cities), and given an itinerary of an array of cities, calculate the total distance. How would you teach students about solving this problem programatically (using a 2D array)?

Recording of how we taught concepts involved in the solution to this problem can be found here: https://youtu.be/Vrukh2LKvbE (from 00:00 to 43:48). The assumption is that the previous lectures already covered the basics of manipulating 2D arrays. This lecture is composed of various forms of illustrations:

- Using a slide show, review the problem at a high level without mentioning any code. (00:00 – 01:44)
- Using the tablet, illustrate how program variables are declared and manipulated. (1:44 – 6:47)
- Using the Android Studio IDE, illustrate a use-case for the calculator (with no errors). (6:47 – 10:06)
- Using a first starter page on the tablet, containing the main code of calculation, illustrate for the given use-case and how each line of code is executed, by visualizing how variables are initialized and manipulated. (10:06 – 27:03)
  Figure 2 (p4) compares the "clean" starter page and the annotated page at the end of this illustration.
- Using a second starter page on the tablet containing code handling errors, illustrate how the flow execution would branch differently, again by visualizing how variables are manipulated accordingly. (27:03 – 37:57)
- Starting with a blank page on the tablet, respond to a student's confusion as to how the two parts of the code work. Here the illustration is meant for reviewing the critical control structure from scratch. (37:57 – 43:48)

Remark. The above teaching pattern—choreographing slide show, code demo on a programming IDE, tablet illustration with a starter page, and answering questions or making additional remarks using tablet illustration with a blank page—is sufficiently general and may thus be applied to teaching many other topics.

In order for students from CS1, CS2, and CS3 to access recorded illustrations and notes for their self-study, even after the completion of the courses, we have maintained a public site of recorded lectures organized by topics, and containing hyperlinks to the recording on YouTube, PDF notes of tablet illustrations, example source code, and slides. Examples on teaching other complicated topics in all three courses can be found on this lectures page: https://www.eecs.yorku.ca/~jackie/teaching/lectures/index.html.

5 EVALUATIONS

5.1 Improvement on Performance

Table 1 (p3) presents data on students' performance on questions related to the various complex ideas:

(1) Subcontracting (Inheritance of Contracts in Descendant Classes)
(2) The Visitor Design Pattern
(3) Genericity
(4) Formal Verification of Software (Proofs of Loop Correctness and Termination)
(5) Object-Oriented Programming (Inferring Classes, Attributes, and Methods from a Given API Tester)

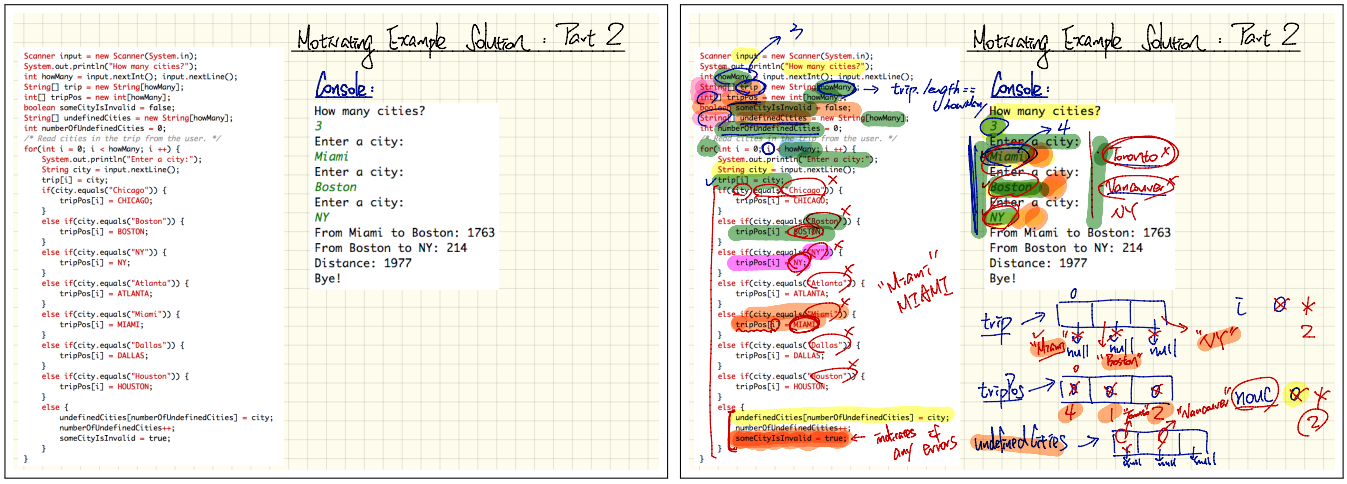
Topics (1) to (4) were taught in CS3[2] by the author in both Summer 2015 (where the proposed technique was not adopted) and Fall 2017 (where the proposed technique was adopted), and we compare the percentage values of students' scores on the same set of final exam questions. Topic (5) was taught in CS1 by the author in both Spring 2017[3] (where the proposed technique was not adopted) and Winter 2018[4] (where the proposed technique was adopted), and we compare the percentage values of students' scores on the same set of computer test questions.

Course	CS3 (SU15)	CS3 (F17)
Proposed Technique Adopted?	No	Yes
Class Size	49	80
Topic	**Student Average Scores**	
Subcontracting	51.63%	54.81%
Visitor Pattern	51.33%	58.33%
Genericity	63.27%	67.00%
Formal Verification of Software	63.62%	63.17%
Course	**CS1 (SP17)**	**CS1 (W18)**
Proposed Technique Adopted?	No	Yes
Class Size	38	190
Topic	**Student Average Scores**	
Object-Oriented Programming	42.97%	56.4%

Table 1: Comparison of Student Performance

Table 1 (p3) indicates that our proposed teaching technique has a positive impact[5] upon students' performance on complex topics, particularly the visitor design pattern (a 16.64% improvement from 51.33% to 58.33%) and programming with classes and objects (a 30% improvement from 42.97% to 56.4%).

[2]Both instances had a large project. Before the project, the earlier instance of CS3 in 2015 had a single assignment, whereas the one in 2017 had six.
[3]This topic was taught at the author's previous institution, in a course whose curriculum overlaps with that of CS1.
[4]Due to a labour disruption at our institution in Winter 2018, this test occurred during the remediation period, where only a subset of the students participated.
[5]Although the results of an independent t-test showed that there was no significant difference between the two groups, in Subcontracting ($t(127) = .77$, $p > .05$), Visitor Pattern ($t(127) = 1.81$, $p > .05$), Genericity ($t(127) = .72$, $p > .05$), and Formal Verification of Software ($t(127) = .13$, $p > .05$), there was a significant difference between the two groups in Object-Oriented Programming ($t(226) = 2.29$, $p < .05$).



(a) Before Annotations Began (11:02) **(b) After Annotations Ended (27:03)**

Figure 2: Illustrating a Code Fragment on a Drawing Tablet (11:02 – 27:03 of https://youtu.be/Vrukh2LKvbE)

5.2 Student Evaluations

The anonymized online course evaluations of the three context courses, which contain both numerical ratings and student comments[6], indicate that the in-class instruction, which relies on the proposed technique of this report, is effective. The data are representative due to the high response rates:

Course	CS1	CS2	CS3
Response	58.09% (219/377)	58.42% (59/101)	85.73% (70/82)

In this section, we present the numerical results (on a 7-point scale[7]) for those questions that are relevant to the effectiveness of teaching and learning:

Q1: The course helped me grow intellectually.
Q2: The course learning outcomes were clearly stated and achieved in the course.
Q3: The instructor conveyed the subject matter in a clear and well-organized manner.
Q4: The instructor helped me understand the importance and significance of the course content.
Q5: Overall, the instructor was an effective teacher in this course.

Table 2 (p4) summarizes the responses grouped as: Agree (denoting the percentage of responses > 4), Disagree (denoting the percentage of responses < 4), and Neutral (denoting the percentage of responses = 4). Table 2 clearly indicates that students value the instructor's method of teaching as proposed in this paper. Questions 1, 2, and 4 explicitly address students' views on their learning outcomes and indicate that the approach is highly effective in helping them achieve the course learning outcomes and grow intellectually. Although the evaluation results for **Q3** – **Q5** are not available for CS1[8], their results may be anticipated as similar to those of CS2 and CS3, extrapolated from observing that the majority of essay results of the three courses are positive about the teaching instructions.

		Q1	Q2	Q3	Q4	Q5
CS1	agree	82.33	90.6	not available		
	neutral	9.02	4.51	not available		
	disagree	7.15	4.14	not available		
CS2	agree	91.53	98.3	100	98.3	96.61
	neutral	6.78	0	0	0	1.69
	disagree	1.69	1.69	0	1.69	1.69
CS3	agree	80	80	94.28	98.3	90
	neutral	1.43	11.43	2.86	0	2.86
	disagree	18.57	8.58	2.86	10.0	7.25

Table 2: Numerical Evaluation Results: Distribution

Furthermore, Table 3 (p5) suggests that answers to all questions have a high median value, and have a mean value that is consistently[9] higher than that of the department and faculty.

6 ADOPTING THE APPROACH

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

Install the following software programs on your teaching computer (e.g., a MacBook): **1)** a presentation program (e.g., any PDF reader, PowerPoint reader) for your slides; **2)** a programming IDE

[6]Some student comments are quoted in Section 7.
[7]**7** for "Strongly Agree", **6** for "Agree", **5** for "Somewhat Agree", **4** for "Neither Agree nor Disagree", **3** for "Somewhat Disagree", **2** for "Disagree", and **1** for "Strongly Disagree".
[8]Due to a recent labour disruption at our institution, **Q3** – **Q5** were excluded from the course evaluation, preventing instructors supporting it by not continuing with classes.
[9]The only exceptions are the mean values of **Q1** and **Q2** for CS3, which may be explained by the essay results, where a good number of students do not appreciate the chosen language for teaching design that is not at the same time a popular implementation language in the industry.

		Q1	Q2	Q3	Q4	Q5
CS1	mean	**5.76**	**5.97**	not available		
	median	6.0	6.0	not available		
	std. dev.	1.41	1.13	not available		
	dep. mean	5.3	5.48	not available		
	fac. mean	5.48	5.65	not available		
CS2	mean	**6.32**	**6.34**	**6.74**	**6.59**	**6.51**
	median	7.0	7.0	7.0	7.0	7.0
	std. dev.	1.11	0.99	0.55	0.65	0.86
	dep. mean	5.63	5.74	5.89	5.78	5.89
	fac. mean	5.66	5.8	5.87	5.82	5.88
CS3	mean	5.41	**5.67**	**6.39**	**6.10**	**6.23**
	median	6.0	6.0	7.0	7.0	7.0
	std. dev.	1.97	1.53	1.07	1.70	1.33
	dep. mean	5.63	5.74	5.89	5.78	5.89
	fac. mean	5.66	5.8	5.87	5.82	5.88

Table 3: Numerical Evaluation Results: Mean and Median

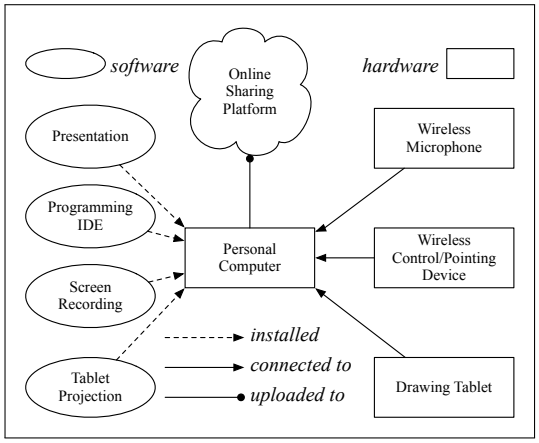


Figure 3: Adopting the Approach: Schematic View

as applicable to your course (e.g., Android Studio, Eclipse); **3)** a screen recording program (e.g., the free Active Presenter [10][10]) 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).

Upon arriving in the classroom, connect the following hardware to your teaching computer: **1)** a wireless microphone (e.g., Revolabs xTag [15]) using a USB cable; **2)** a wireless control or pointing device (e.g., a wireless or bluetooth mouse, trackpad, keyboard, laser pointing device) for showing slides, typing code in an IDE, or switching between programs; and **3)** a drawing tablet (e.g., iPad Pro) installed with an app for annotations (e.g., GoodNotes, Notability). For **3)**, a wire connection to the USB port is recommended for a stable connection throughout the class. 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.

[10] Another lecture capturing system such as Panopto and TechSmith may also work.

When ready to start your lecture: **1)** wear the wireless microphone connected to your teaching computer (and optionally another microphone connected to the classroom speaker); and **2)** start the screen recording program and choose the connected wireless microphone as the input device.

When the lecture 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), and publish the link to students. The annotation app on your drawing tablet should allow you to export the annotated notes (e.g., Figure 2a, p4) as an PDF file.

7 REFLECTION

In this section we share our reflections on the proposed approach, after adopting it for teaching the three courses (Section 2).

Drawing Tablet vs. Blackboard/Whiteboard. Often students sitting at the rear or sides of a class room find it difficult to copy contents of the front blackboard/whiteboard into their notes, let alone comprehend the concepts and processes being illustrated. Instead, projecting contents to a high, centred screen allows students not to miss any parts of the instructor's presentation. More importantly, the use of an annotation app allows the instructor to save time on copying, e.g., code fragments, in order to start a new discussion. Instead, a starter page on the tablet can be preset and launched at the appropriate time. 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.

Essay results of student evaluations from all three context courses confirm the effectiveness of using a drawing tablet: e.g.,

- "Lectures are well planned and use of iPad with slides is a great way to explain things." [CS1]
- "[Best things about the course are] Using iPad notes to explain the logic and to trace the certain parts of code. " [CS1]
- "the teaching style of instructor is awesome, he uses an Ipad instead of black board and record every lecture, which is very helpful. if this kind teaching style is used by other instructor then student will pay more attention to teacher then copying notes from blackboard and doing multitasking." [CS2]
- "Great supplementary materials (recordings, lecture notes, ipad notes, etc..)" [CS3]

Drawing Tablet vs. Slide Animations. Animated slides have the great advantage that information can be revealed at a planned time. However, when teaching complicated ideas, it is often not effective to encode all details in animations, since the order of animations is *static*, and the instructor may not be able to account for how students actually understand the concepts dynamically in class. Instead, with the use of a drawing tablet, the instructor can teach more effectively by creating a starter page, and then gradually adding annotations as illustrated in Section 4. This is essentially a *dynamic* way to control the pace and level of details that the instructor judges appropriate for how the current class is understanding the materials.

Review of Lectures. We strongly believe that the most valuable component of in-class instruction is the *dynamic* illustrations of difficult concepts. Despite how effective such illustrations are, we

would never expect students to fully comprehend the concepts from in-class presentation alone simply because they are hard and naturally require repetitions. As a result, being able to record all transitions among the slide show, code demos, and tablet illustrations is invaluable for students to review the concepts after class. Such review[11] also helps them reflect on the materials and ask thoughtful questions. Furthermore, the proposed approach also allows students who miss classes for legitimate reasons (e.g., medical, family) to catch up with the course content at their own pace. To facilitate students' selection of parts of a lecture recording which they are interested in reviewing, we may add links to the various timings of starting critical examples or concepts.

A Suitable, Transferrable Teaching Pattern. How would the teaching and learning experience be different if the proposed approach, particularly the interactive experience of tablet illustrations, was not adopted for teaching the 2D array solution (Section 4)? Our previous experience of teaching concepts with a similar or higher level of difficulty, without using the proposed approach, was that we had to rush through parts that really need the most detailed, in-depth illustrations, primarily because it was time consuming to copy the starting code fragments onto the board. Even if such illustrations were performed, students would not be able to review them after class. Therefore, with the ability to record illustrations on a drawing tablet, the teaching pattern as observed in Section 4 may be generalized as follows:

- Use slide show to present the general problem to be solved.
- Use a programming IDE to demonstrate what is ultimately expected from the final (e.g,. software) product.
- Pre-set a list of starter pages on the drawing tablet, each containing selected code fragments or formulas, then annotate them to gradually build towards the solutions or conclusions.
- Answer students' questions by starting a blank page on the drawing tablet, and build up the answers there from scratch.

Some items (except tablet illustrations) may be omitted, and the order of choreographing these components may be adjusted according to the subject being taught.

Required Preparation. The instructor should determine what concepts/examples they will illustrate in class, and then create the starter pages on the drawing tablet accordingly. Compared with simply writing them "on they fly" in class, the instructor may take more time carefully planning the layout of each page. But most valuably, these starter pages, once created, may be *elaborated* over time and later *reused* for teaching the same or similar subjects.

Complexity of Integration. The current practice of the proposed approach requires us to manually assemble an array of equipment (Section 6). A better solution is to have the classroom podium computer being setup according to the schematics (Figure 3, p5).

8 RELATED WORKS

Our proposed teaching technique requires the careful setup of starter artifacts (e.g., code fragments, figures, writings) on a drawing tablet for in-class illustrations, which are recorded for students to review after class. Our technique is suitable for teaching complex computational thinking [2, 5, 6, 14] through carefully planned illustrations on a drawing tablet which, based on our experience, are perceived as less difficult and boring by students compared with in-class instruction that heavily relies on slide shows. Although there are other systems allowing the instructor to handwrite on slides projected from a tablet PC (e.g., [1]), our approach also allows other desktop activities (such as demonstrating on a programming IDE) to be integrated into the lecture instruction and recording.

The platform offering the most similar support is the MIT Open Course [8], but the posted contents there are limited to filming of instructors and the projector screen (as opposed to desktop activities of the instructor's computer) and in-class illustrations on a blackboard. Setting up the starting ground for such illustrations on a conventional in-class blackboard "on the fly" takes up class time, whereas in our proposed approach, the instructor can carefully set up the starter artifacts (e.g., code fragments, figures, writings) on the drawing tablet and plan their lectures accordingly.

Also relevant is the Stanford Online [13], which is designed for online distance learning. 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. On the other hand, we have also adopted the proposed technique to prepare tutorial videos as detailed pre-study materials for lab assignments. Our students expressed in their written comments that these tutorial videos, with detailed and in-depth illustrations on a drawing tablet, are valuable study materials for completing lab assignments and preparing for tests. For example, for *CS1*, a tutorial series (21 videos, 6 hours) is made available on day one of the course on developing, from scratch, an Android Mobile app for calculating the BMI (Body Mess Index), with a clear separation among the model, the view, and the controller. Similar to Stanford Online are the online course repositories such as Coursera [3] and Udemy [12], but these are meant to be commercial, unlike the proposed approach as well as MIT Open Course and Stanford Online. Moreover, the use of intensive tablet illustrations is also not typical in these commercial courses (e.g., an introduction to Android app development).

9 CONCLUSION

We describe the use of a drawing tablet to conduct in-depth and detailed illustrations of complex ideas, and to record all in-class desktop activities on the instructor's computer (including slide show, code demonstrations, tablet illustrations). Comparison of student performance on complex topics indicates a positive impact of our approach. Student evaluation results also indicate that this approach is effective for helping students achieve the expected course learning outcomes.

The proposed approach was only adopted in three undergraduate CS courses so far. Nonetheless, given **1)** the total number (344) of students participating in the online course evaluation; **2)** the vast majority of (numerical and essay) evaluation results concerning the effectiveness of the instruction being positive; and **3)** the wide range of topics being covered in these courses, we believe that this experience report would benefit colleagues who wish to improve their teaching of complex (programming, design, or abstract) ideas.

To reaffirm the effectiveness of the approach, we will adopt it in teaching courses involving more abstract theories of computation.

[11] According to the YouTube statistics, students in all three courses spent a decent amount of time reviewing the recorded lectures: each student in CS1 spent an average time of 213 minutes, 740 minutes for CS2 students, and 871 minutes for CS3 students.

REFERENCES

[1] Richard Anderson, Ruth Anderson, Beth Simon, Steven A. Wolfman, Tammy VanDeGrift, and Ken Yasuhara. 2004. Experiences with a Tablet PC Based Lecture Presentation System in Computer Science Courses. *SIGCSE Bull.* 36, 1 (March 2004), 56–60. https://doi.org/10.1145/1028174.971323

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

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

[4] C. A. R. Hoare. 1969. An Axiomatic Basis for Computer Programming. *Commun. ACM* 12, 10 (Oct. 1969), 576–580. https://doi.org/10.1145/363235.363259

[5] 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 (ITiCSE 2018)*. ACM, New York, NY, USA, 69–74. https://doi.org/10.1145/3197091.3197120

[6] 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

[7] Bertrand Meyer. 1997. *Object-oriented Software Construction (2Nd Ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[8] Masschusetts Institute of Technology. [n. d.]. MIT Open Courseware. https://ocw.mit.edu/index.htm

[9] D. L. Parnas. 1972. On the Criteria to Be Used in Decomposing Systems into Modules. *Commun. ACM* 15, 12 (Dec. 1972), 1053–1058. https://doi.org/10.1145/361598.361623

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

[11] SMART. [n. d.]. SMART Board for Education. https://smarttech.com/

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

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

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

[15] Revolabs xTag. 2007. Wireless Microphone System. Model 02-DSKMAN-DPP-11.