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

Computer Science Education Research Conference
CSERC’19 / Nov 18 / Larnaca, Cyprus

Chen-Wei Wang
York University, Toronto, Canada
Challenges of Undergraduate Teaching

1. **complex computational thinking**: limited prior exposure
   - e.g., OOP: class associations and loops [paper]
   - e.g., OOP: polymorphic collection and dynamic binding [talk]

2. **weekly laboratories**: lectures $\not\Rightarrow$ pre-requisites
   - Lab assignment are important opportunities for students to achieve the intended learning outcomes.
   - Instructors should provide in-depth remarks and illustrations on examples, reflecting their insights into the subjects, but . . .
     - fixed lecture hours $\not\Rightarrow$ logical decomposition of topics
     - limited lecture hours $\not\Rightarrow$ thorough, uninterrupted discussion
How to Help this Frustrated Student?

Frustrated Student: 
I *did attend* classes 
but *could not complete the weekly lab assignments.*
Motivating Question

How can we make the in-depth and thorough *illustrations* accessible to students for their *self-paced study* outside the classroom so as to help them complete the *lab assignments*?
Contribution: Creating Effective Tutorials on Complex Ideas

A technique for

○ *Recording illustrations* of *complex ideas* on a *drawing tablet*.

  ● *Pre-recording* preparation of *starter artifacts* (e.g., code fragments, diagrams)

  ● *Frequent* and *heavyweight* *annotations*

○ *Allowing students to* *study* *outside* class at their *own pace*

Let’s illustrate the technique using a short *tutorial* on *polymorphism* and *dynamic binding* in OOP.
class Course {
    private String title;
    private double fee;

    Course(String title, double fee) {
        this.title = title;
        this.fee = fee;
    }

    String getTitle() {
        return this.title;
    }

    double getFee() {
        return this.fee;
    }
}
class Student {
    private String name;
    private Course[] courses;
    private int noc; /* number of courses */

    Student(String name) {
        this.name = name; this.courses = new Course[10];
    }

    String getName() { return this.name; }

    void register(Course c) { this.courses[noc] = c; this.noc ++; }

    double getTuition() {
        double base = 0;
        for(int i = 0; i < noc; i ++) {
            base += this.courses[i].getFee();
        }
        return base;
    }
}
class ResidentStudent extends Student {
    ResidentStudent(String name) {
        super(name);
    }

    private double premiumRate;

    double getPremiumRate() {
        return this.premiumRate;
    }

    void setPremiumRate(double r) {
        this.premiumRate = r;
    }

    double getTuition() {
        double base = super.getTuition();
        return base * premiumRate;
    }
}
class NonResidentStudent extends Student {

    NonResidentStudent(String name) {
        super(name);
    }

    private double discountRate;

    double getDiscountRate() {
        return this.discountRate;
    }

    void setDiscountRate(double r) {
        this.discountRate = r;
    }

    double getTuition() {
        double base = super.getTuition();
        return base * discountRate;
    }
}
class StudentManagementSystem {
    Student[] students;
    int nos; /* number of students */

    public StudentManagementSystem() {
        students = new Student[10000];
    }

    void add(Student s) {
        this.students[this.nos] = s;
        this.nos ++;
    }

    Student[] getStudents() {
        Student[] ss = new Student[this.nos];
        for(int i = 0; i < this.nos; i ++) { ss[i] = this.students[i]; } 
        return ss;
    }
}
public class SMSTester {
    public static void main(String[] args) {
        Course eecs2030 = new Course("Advanced OOP", 1000.0);
        Course eecs3311 = new Course("Software Design", 1000.0);
        ResidentStudent heeyeon = new ResidentStudent("Heeyeon");
        heeyeon.setPremiumRate(1.25);
        heeyeon.register(eecs2030);
        heeyeon.register(eecs3311);
        NonResidentStudent jiyoon = new NonResidentStudent("Jiyoon");
        jiyoon.setDiscountRate(0.75);
        jiyoon.register(eecs2030);
        jiyoon.register(eecs3311);
        StudentManagementSystem sms = new StudentManagementSystem();
        sms.add(heeyeon);
        sms.add(jiyoon);
    }
}

Exercise 1: How do L14 & L15 result in a polymorphic array.
Exercise 2: Add code to output the tuition due for students.
Demo Tutorial: Expected Console Output

- Let’s first see how the expected output look like!

  Heeyeon should pay $2500.0
  Jiyoon should pay $1500.0

- Given:

  ```java
  class StudentManagementSystem {
      Student[] students;
      ...
  }
  ```

  How can our code ensure that the tuition of:
  
  - 1st *resident* student is calculated using *premium* rate.
  - 2nd *non-resident* student is calculated using *discount* rate.

- Let’s code this up!
A Pattern for Tutoring Complex Ideas

- I just demonstrated a *tutoring pattern*, choreographing:
  - **Specify the Problem**: Slide Show and/or Programming IDE
  - **Sketch the Solution**: Drawing Tablet
  - **Develop the Solution**: Programming IDE
  - **Discuss the Solution**: Drawing Tablet

- When the *drawing tablet* is used:
  - *Annotate* on starter pages to explain *critical steps* in the solution. e.g., *starter* page vs. *annotated* page in the example lecture

- More examples:
  - Paper: teaching an OO programming pattern using primitive arrays
  - My lectures page (with links to various tutorials):
    - [https://www.eecs.yorku.ca/~jackie/teaching/lectures/index.html](https://www.eecs.yorku.ca/~jackie/teaching/lectures/index.html)
Contribution: An Approach for Creating Effective Tutorials

- Instructor
  - Computer Desktop Screen
    - Slide Show
    - Code Demos on Programming IDE
    - Illustrations on Drawing Tablet

- Outside-class, pre-lab
  - Recording
  - Illustration Notes
  - Source Code
  - Present
  - Information flow
  - Recorded & uploaded
  - Re-iterated on demand

- Outside-class
  - Students

LASSONDE
SCHOOL OF ENGINEERING
14 of 23
Study Resources: Video Playlist

York Lassonde EECS1021 Java Tutorial

46 videos • 35,920 views • Last updated on Mar 11, 2019

Public

iPad Notes here:
https://www.eecs.yorku.ca/~jackie/teaching/tutorials/notes
/EECS1021%20Tutorial%20on%20Java.pdf

Jackie Wang

EECS1021 Java Tutorial 41
Jackie Wang
1:23:57

EECS1021 Java Tutorial 42
Jackie Wang
1:18:08

EECS1021 Java Tutorial 43
Jackie Wang
44:27

EECS1021 Java Tutorial 44
Jackie Wang
30:21

EECS1021 Java Tutorial 45
Jackie Wang
44:07

EECS1021 Java Tutorial 46
Jackie Wang
28:57
Tutorial on Object-Oriented Programming in Java
Teaching Context

Proposed approach adopted in *undergraduate teaching*:

- **7 iterations** of four courses [1st-, 2nd-, 3rd-year]
- Created **12** series of **148** tutorial videos (≈ **59.5** hours)
- Tutored **1,295** students
  - e.g., *Java Programming from Scratch*
    - variables, assignments
    - if-statements, loops, arrays
    - classes, attributes, methods, objects, aliasing
  - e.g., *OOP for Developing Android Mobile Apps*
    - Model-View-Controller
  - e.g., *Developing a Birthday Book Application in Java*
    - multiple classes
    - complex loops

Nonetheless, the proposed approach is sufficiently general for tutoring any complex idea.
Reflections

- **Instructor’s Efforts**
  - **Starter Pages**: What concepts/examples should be illustrated?

- **Drawing Tablet vs. Blackboard/Whiteboard**
  - **Time Effectiveness**: Starter pages let us get straight to the point.
  - **Reusability**: Starter pages may be elaborated and reused.

- **Drawing Tablet vs. Slide Animations**
  - **Flexibility**: Dynamic control of the pace and level of details w.r.t. the comprehension level.
    - e.g., *starter* page vs. *annotated* page in the example lecture

- **Review of Tutorials**
  - **Repetition**: Even effective illustrations take repetitions to achieve full comprehension.
Beyond this talk... 

- Read my paper!
  - Adopting the Approach
  - Evaluation: Students’ Perception
  - Evaluation: Improvement on Students’ Performance
  - Comparison with Related Works

- Similar approach adopted for delivering effective lectures:


Questions?
Teaching Challenge: Big Classes
Adopting the Approach

- **software**
  - Presentation
  - Programming IDE
  - Screen Recording
  - Tablet Projection

- **Online Sharing Platform**

- **hardware**
  - High-End Studio USB Microphone
  - Drawing Tablet

- **Personal Computer**

- **installed**
- **connected to**
- **uploaded to**
Challenges of Undergraduate Teaching
How to Help this Frustrated Student?
Motivating Question
Contribution:
Creating Effective Tutorials on Complex Ideas
Demo Tutorial: Recall from Last Tutorial (1)
Demo Tutorial: Recall from Last Tutorial (2)
Demo Tutorial: Recall from Last Tutorial (3)
Demo Tutorial: Recall from Last Tutorial (4)
Demo Tutorial: Recall from Last Tutorial (5)
Demo Tutorial: Console Tester
Demo Tutorial: Expected Console Output
A Pattern for Tutoring Complex Ideas
Contribution: An Approach for Creating Effective Tutorials

Study Resources: Playlist

Study Resources: iPad Notes

Teaching Context

Reflections

Beyond this talk…

Teaching Challenge: Big Classes

Adopting the Approach