An Annotation Application for Grading Papers

Notetaking Applications in Computer Science and their Connection with Different Notetaking Strategies and Theories of Learning in Computer Science Education

A Thesis Prospectus
In STS 4500
Presented to
The Faculty of the
School of Engineering and Applied Science
University of Virginia
In Partial Fulfillment of the Requirements for the Degree
Bachelors of Science in Computer Science

By
Andrew Song

Fall 2022

ADVISORS

Kent Wayland, Department of Engineering and Society

Rosanne Vrugtman, Department of Computer Science

General Research Problem: Improving Computer Science Education Through Notetaking Applications

How can notetaking applications be created to cater towards computer science education and learning?

Just like any other subject, building a foundation of knowledge in computer science begins with learning in the classroom, and a big part of learning in the classroom is being able to take notes effectively. More and more students use digital notetaking platforms and applications to take notes in the classroom. Despite the convenience, there are clear limitations to taking notes digitally. At the same time, digital notetaking also offers unique opportunities and methods of taking notes. The limitations and opportunities presented by digital notetaking are especially clear when it comes to taking notes in computer science classes. For example, drawing out computer program representations can be quite difficult to do digitally. On the other hand, only digital platforms provide the ability to create animations to represent and visualize programs.

Thus, I am very interested in answering the question of how we can address these limitations of digital notetaking, while also taking advantage of these opportunities of digital notetaking, to create applications that are catered towards computer science education and learning. To explore answers to this question, I aim to expand upon my own technical work of creating an annotation application for grading papers and combine that with my STS research where I explore various computer science notetaking applications and their use of notetaking strategies that are prevalent today and examine how they fit into theories of learning in computer science. I believe that my annotation application shares similar core principles and functionalities behind notetaking applications while also helping to explore practical user interface design for taking notes. On the other hand, I believe that my STS work offers insight into the ideals and theory that go into making a notetaking application that will be optimized for learning computer science subjects.

An Annotation Application for Grading Papers

How can an annotation application be created in order to grade papers while reducing the amount of time spent on grading?

Being able to save time is one of the most important benefits that has come with the rise of digital technologies in the classroom over the years. This is particularly true for teachers and professors when it comes to grading. The emergence of digital assessment platforms, in particular, has automated the grading process, requiring little to no human input to grade tests and assignments. While many applications have been created to simplify and automate the process of grading assignments that have pre-determined answers, a different approach is required when it comes to simplifying grading for assignments that have open-ended answers, such as papers. Because of the subjectivity involved in grading papers, it is difficult to create a system that can grade and assess papers. Thus, other approaches besides automated feedback are required to try and reduce the amount of time it takes to grade papers.

Some current approaches to reducing grading time for open-ended questions involve the use of machine learning to process assignments and give them a grade. For example, a team of programmers created an unsupervised learning program that is able to take an ungraded response to an unseen question and give a grade on that assignment (Chen et al., 2010). However, even if this method was to be “generally reliable” and save time, it is easy to imagine that graders would feel uncomfortable having virtually no input on the feedback given to their students. As a result, they are likely to go through the paper and add their own feedback while also trying to verify and check the program feedback, ultimately not saving much time. Another approach uses machine learning for a different purpose. The JANE system uses machine learning to learn annotating patterns and automatically generate potential annotations for the user to select from while annotating (Tomanek et al., 2007). This particular approach not only saves time during the annotation/grading process, but also attempts to save time by learning the feedback that is being repeated. However, graders may often rather sacrifice the annotation pattern learning to be able to precisely write their own comments that they want to give.

Our approach to make this grading process faster is to create an interface that speeds up the process of writing feedback by taking advantage of the idea that professors oftentimes find themselves giving similar feedback over and over throughout their grading. Our application first allows graders to upload a rubric for a given assignment, parsing all the different comments/feedback on the rubric and points taken off for those particular comments. For example, a rubric may contain the comment: “Your introduction went into too much detail about the problem you are describing” for which 0.5 points would be taken off. Once the rubric has been fully processed, our application incorporates those into an annotating interface in which graders can quickly select feedback to add from the pre-set list of comments. Our research team designed the web application with a javascript frontend (with the React framework) and a python backend (with the Django framework). The database used in the application was a PostgreSQL database. The application scrapes a submission site to display student submissions in an annotation interface from which graders can add feedback and annotate the papers. After about a year of work, the result was a prototype application that was tested briefly by a professor and the teaching assistants to grade papers in a real course.

This work presents several foundational ideas and implementations that would be useful in creating a notetaking application to aid in CS education. Firstly, the idea that grading often involves writing and typing the same things again and again also applies to notetaking; many notes and annotations can often be the same/similar. Thus, this functionality is something that would be useful while pursuing the research project. Additionally, I believe that the user interface created in my technical work could serve as a foundation for the interface of application built in future annotating or notetaking applications.

Notetaking Applications in Computer Science and their Connection with Different Notetaking Strategies and Theories of Learning in Computer Science Education

How do computer science notetaking apps built today make use of computer science notetaking strategies to fit various theories of learning in computer science education?

Because traditional notetaking methods can fall short in some respects when learning computer science subjects, many digital notetaking applications have been created and geared specifically for learning computer science subjects. To be able to evaluate these notetaking applications, one approach is to examine popular notetaking applications and analyze how they make use of certain notetaking strategies to reinforce, or fail to reinforce, various theories of learning in computer science. For example, we may evaluate a popular notetaking app called Obsidian to determine whether one of the notable notetaking strategies that the app offers—support for graph creation to aid visualization and organization—fits or does not fit under a well-known theory of computer science education called constructivism. Without understanding the connections between notetaking apps and the theories of learning, it becomes difficult to know not only which notetaking methods are truly effective, but also to know how to develop and improve notetaking strategies when learning computer science subjects. Thus, I would like to find out some current notetaking strategies for computer science subjects along with prevalent theories of how students learn computer science effectively and analyze the ways in which popular notetaking applications for computer science make use of certain notetaking methods to reinforce certain theories of learning while ignoring others.

Background

Ever since computer science became a more formal discipline in education in the 1960s, much research and study in computer science education has led to a number of theories regarding how students learn computer science subjects effectively. These theories clash in some ways while mixing in other ways and are topics of great interest in computer science education. Interestingly, different groups of people tend to find different theories more relevant and applicable to education. Software engineers, researchers, and students are some groups that have beliefs on this topic. Ultimately, in my research, I would like to explore why certain groups prefer a particular theory of learning and the influence of these various groups on today's note taking applications in computer science by examining the notetaking strategies of these applications and seeing how they fit into certain theories of learning.

Literature on Theories of Learning and Social Factors Influencing Theories of Learning

To begin exploring the question that has been proposed for research, one must first understand some of the primary theories of learning in computer science. Constructivism is one of the most popular theories in computer science education. At the highest level, the main idea behind this theory is the idea that "students construct knowledge rather than merely receiving and storing knowledge transmitted by the teacher" (Ben-Ari, 2001). Another popular theory is cognitivism. This theory focuses on the idea that cognitive processes like induction, deduction, pattern recognition, etc. are the keys for learning and understanding (Taylor et al., 2013). Other theories focus on the social aspects of education. For example, social construction is a theory that focuses on "how the individual interacts with a community in developing their understanding" (Machanick, 2007). Similarly, the situated learning theory argues that a "community of practice" is needed for deep learning of material (Ben-Ari, 2004).

Given these theories of learning, one question that arises is: which theories are more prevalent and why are certain theories pushed more than another? One reason is the innate beliefs of different groups of interest. In particular, software engineers tend to favor the constructivism theory because of how it applies to their day-to-day work. Because of the huge amount of knowledge that exists in the field, it is impossible to try and learn everything by being taught the material. Instead, software engineers often find themselves constructing their own knowledge while developing applications, manuals, interfaces, and so on. On the other hand, researchers who are generally theory-oriented tend to favor the cognitive method of learning (Ben-Ari, 2001).

Recently, the rise of virtual learning in schools due to COVID has resulted in a shift in theories of learning. Without face-to-face interaction between students and teachers, the importance of constructivism has been more clearly realized. Students are more often required to construct their own knowledge for learning as it became more difficult to receive in-person lectures from teachers and professors. As a result, companies like Google and Microsoft adapted their services fit these needs by making them more user-friendly for students and teachers

(Davenport et al., 2022). This serves as an example of companies adjusting to social factors causing shifts in learning theories.

Methods and Evidence

One primary source that I intend to use in my research are reviews for various notetaking applications, along with update history of these applications. Reviews give an idea of what various groups of people are thinking, and what they want in notetaking applications, and application update histories reflect the companies responding to these ideas. Additionally, I plan to research discussions, blogs, and interviews with developers of the applications that I look into. These could help to understand reasoning behind the app's features and what theories of learning they may fit into.

Other evidence that I intend to use are journals, surveys, and research papers. These provide information on the specific notetaking strategies and theories of learning. Further, they detail the state of the sociotechnical system which my research looks into. Surveys of students can gather student opinion of notetaking methods and learning theory, for example.

Conclusion

From my STS research, I hope to gain an understanding of various notetaking strategies along with different theories of learning in computer science and see how companies have built applications to incorporate these strategies and fit different theories of learning. I believe that this would be beneficial in answering my general research question by providing an understanding of how currently existing popular applications try to aid learning in computer science. On the other hand, my technical research presents groundwork and ideas/implementations for the functionality and interface of the proposed research application. Together, these two research projects may serve as a foundation for future works that aim to create notetaking applications that cater towards learning in computer science education.

References

Ben-Ari, M. (2001). Constructivism in Computer Science Education. *Journal of Computers in Mathematics and Science Teaching, 20*(1), 45-73. Association for the Advancement of Computing in Education (AACE).

Ben-Ari, M. (2004). Situated Learning in Computer Science Education. *Computer Science Education*, *14*(2), 85-100. DOI: 10.1080/08993400412331363823

Chen, Y. Y., Liu, C. L., Lee, C. H., & Chang, T. H. (2010). An unsupervised automated essay-scoring system. *IEEE Intelligent systems*, *25*(5), 61-67.

Davenport, J., Dawson, A., Day-Lewis, D., Delgado, R., & Kelt, T. (2022). *Constructivism in 21st-Century Online Learning*. Open Journal of Information Systems.

Machanick, P. (2007) A Social Construction Approach to Computer Science Education, *Computer Science Education*, *17*(1), 1-20, DOI: 10.1080/08993400600971067

Taylor, E., Breed, M., Hauman, H., & Homann, A. (2013). *Choosing Learning Methods Suitable for Teaching and Learning Computer Science.* International Association for the Development of the Information Society (IADIS).

Tomanek, K., Wermter, J., & Hahn, U. (2007). Efficient Annotation with the Jena ANnotation Environment (JANE). *Proceedings of the Linguistic Annotation Workshop*, 9-16. Association for Computational Linguistics.