IMPLEMENTING SOCIAL LEARNING FOR MORE EQUITABLE COLLABORATION IN INTRODUCTORY COMPUTER SCIENCE EDUCATION

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Seongtaek Lim

August 2019

IMPLEMENTING SOCIAL LEARNING FOR MORE EQUITABLE COLLABORATION IN INTRODUCTORY COMPUTER SCIENCE EDUCATION

Seongtaek Lim, Ph.D.
Cornell University 2019

Despite the collaborative nature of software engineering practice, computer science (CS) education has focused on individual-centric pedagogy in teaching how to program. Existing approaches to teaching it, consequently, often end up ignoring important elements of training in professional programming practice, such as working with teams and solving collaborative challenges in systematic manners.

My research investigates approaches driven by a learning theory to teaching novice students how to collaboratively program. This work first reviews Vygotskian learning theory that informs about how learning happens in social learning contexts. This social development theory says that effective learning occurs in the zone of proximal development where a learner can perform tasks with assist from more knowledgeable others. It also emphasizes the significance of sign systems, tools, and language during the learning process, since they mediate both the direct interaction with others and the indirect interaction with the broader society that embraces the domain of learning.

From the perspectives of student interaction as the source of learning, this work builds and tests a real-time interactive student discussion tool, named MOOCchat, in an online learning platform. The evaluation said that student interaction can improve the performance in quiz solving tasks, even in the isolated online settings. Shifting from the online learning environment to physical classroom settings, this work also reports on an exploratory study to understand how students are currently guided to collaborative learning in programming practice. The results showed that teachers are experiencing a lack of technology support and the standard curriculum for teaching proper ways of collaboration.

To implement the social learning in CS classrooms and meet the needs the prior work identified, this work proposes lesson plans and software tool (GLIDE) built around them. An observational study with the educational intervention in an actual high school classroom with GLIDE found supporting evidence for the learning outcomes and the better balanced contributions from peer collaborators. Finally, this study reports on quantitative evaluations on the consequences of the educational intervention on the students' learning experience, arguing that the proposed approach increases student engagement, psychological ownership in their projects, and perceived fairness in collaboration.

BIOGRAPHICAL SKETCH

Seongtaek Lim's research area falls in the intersection of human-computer interaction (HCI), computer-supported cooperative work (CSCW), and education technology. He builds interactive end-user systems and studies how the software interacts with users and how it changes user behaviors. The software tools he built address various topics including education, social computing, information visualization, and information access and retrieval. He has worked at Fuji Xerox Palo Alto Laboratory and KAIST since he received his B.S. in Computer Science and M.S. in Cognitive Engineering from Yonsei University.

ACKNOWLEDGEMENTS

Looking back my 7 years, I have learned the meaning of being humble through the toughest time of my life. It has been much more valuable lesson from God than any other knowledge I was able to learn.

I wish to thank Tapan Parikh for the guidance, inspiration, and support during my graduate studies. I deeply cherish the time I have spent working with him. I have also benefited greatly from knowing and working with Steven Jackson, Ross Tate, and Rene Kizilcec at Cornell University. In addition, I appreciate all the chances that Marti Hearst at UC Berkeley and Patrik Chiu at FXPAL gave me.

I would not have been able to survive without the encouragement from the numerous colleagues at Cornell University. I would like to thank Sujay Bhatt, Vibhore Vardhan, Anthony Poon, Eugene Bagdasaryan, Neta Tamir, and Samar Sabie. I also can't even forget the support from the folks at UC Berkeley including Adam Calo, Sarah Van Wart, Richmond Wong, and Noura Howell.

I would like to also acknowledge the financial support from many institutions, which made me possible to pursue my goal. The execution of my study was possible in part by the research grants from Google Social Interaction Program and National Science Foundation. I also appreciate the research cooperation from Information Technology High School in Queens, NY and Youth Radio in Oakland, CA.

Finally, I owe the biggest thanks to my family and Gracepoint Church in Alameda, CA for their priceless love and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

There is recently growing interest in understanding how to better educate the next generation of students in computer science (CS). Specific needs for reformed CS education include covering modern topics [27], applying novel pedagogical approaches for students' improved learning in CS [74, 81], and improving inclusiveness of computing education [114]. Such demand for changes has fueled several initiatives of educators, policy makers, and researchers in the field of CS education for next generation students. For example, CS4All of New York City Department of Education (NYCDOE) has incorporated mandatory CS units to ensure that every NYC student at elementary and secondary levels will receive CS education by 2025 [2]. A non-profit organization, such as Code.org [1], also has dedicated to expanding access to CS education to underrepresented populations, providing learning materials and tools on its web platform. Although such endeavors provide us with opportunities and promises to improve the quality and impact of CS education, it is not yet fully discussed and determined on what specific topics to be embedded in typical classroom environment and what pedagogical approaches to be taken in each of the CS courses.

1.1 Review of the Previous Curriculum Development in CS

One of the best ways to answer the question of what and how to teach next generation CS could be looking backward into the history of CS education. According to Curriculum '68 [5] by Curriculum Committee on Computer Science (C3S), one of the earliest reports published on recommended computing curricula currently accessible, basic CS courses for beginners traditionally consisted of

Introduction to Computing, Computers and Programming, Introduction to Discrete Structures, and Numerical Calculus. Other than the general introductory course for larger areas of computing, the rest three courses are suggestive of a huge emphasis on mathematics in CS education at that moment. Particularly, the description of the elementary courses in the report says "certain courses in mathematics are necessary, or at least highly desirable". Among the basic courses listed in the report, Computers and Programming looks rather distinct in that it focuses on learning to use machine-interpretable languages to interact with computer systems. This report still acknowledges the importance of the programming course by stating that the programming course is "intended to lay a foundation for more advanced study in computer science".

A decade later, Curriculum '78 [9] has taken very different perspectives on CS curricula from the previous version [5]. It sets mathematics courses apart from the CS course curriculum, such that only two out of eighteen CS courses, Numerical Analysis and Linear Algebra, in the list directly cover mathematical or algebraic topics. Although this report does acknowledge that mathematical concepts and techniques are crucial in CS education, it justifies elimination of the other mathematics courses by stating "the kind and amount of material needed from these areas for computer science usually can only be obtained, if at all, from the regular courses offered by departments of mathematics for their own majors". In addition, none of mathematics courses and the two algebraic CS courses are included as a prerequisite for entry-level students. Instead, the report recommends students to take four topic areas: programming, software organization, hardware organization, and data structures and file processing. The changes made to the Curriculum '78, compared to Curriculum '68, suggest that CS has begun to be considered as a distinct and separate discipline that solely covers computer systems at that moment. However, this approach has also been criticized for providing potential misunderstanding that computer science is all about programming [94].

In the early 90's, Computing Curricula 1991 [119] shows significant changes in perspectives on CS education from its predecessors. One of the most notable features of this report is that it explains pedagogical principles as well as topics to cover in CS curriculum. Especially, it allocates considerable number of pages to emphasizing the role of programming in general and the importance of lab activities in CS education in a broader context. While it ensures that programming does not stand for the whole software engineering, it argues that novice students should be able to be engaged in the mastery of programming through their coursework, assignments, and lab activities. Furthermore, it demonstrates that lab activities are important for students to experience teamwork projects with other students in order to complete tasks. Although Curriculum '78 also mentioned that team project is a good option as a teaching method ("consideration should be given to the implementation of programming projects by organizing students into programming teams"), but more specific and concrete pedagogical recommendations have been made only in the early 90s.

Since Computing Curricula 2001 [23, 25, 103] has been introduced, CS curriculum has become highly comprehensive and detailed in providing teaching methods and teaching guidelines. The curriculum has been incrementally modified to reflect the rapidly-changing field and emerging topics, such as World Wide Web and its applications, multimedia, and human-computer interaction. From the historical perspective, it seems that CS curriculum has constantly emphasized the importance of exposures to programming activities for students, since its earlier versions. Even though the specific focus on the programming practice has been changed (e.g., from algebraic concepts to networking principles), better elaborated (e.g., concurrency-related issues in 80s and 2000s), or newly introduced (e.g., World Wide Web applications), the significance of programming education has been constantly highlighted in CS curriculum.

To date, little effort has been yet made in developing the practical and specific CS curriculum that can better educate programming skills and knowledge, despite its long-lasting emphasis put on programming education in the previous carricula. One possible implication of the previous curricula is that team projects are highly encouraged and desired for programming practice [25, 103, 119]. None of the previous work, however, has focused on developing concrete and systematic methods for implementing teamwork in programming education, or justifying it by using further analytical and empirical approaches. Therefore, my dissertation work aims to highlight the significance of teamwork in programming education and discuss how to implement it in real world settings. Another point to make based on the history of CS curriculum is that targeted levels of students of interest are steadily expanding. Although the earlier CS curriculum has primarily targeted 4-year undergraduate students, later curricula extended its scope to 2-year college programs [25]. It also suggested that an increasing number of students are expected to get exposed to computing education at the secondary level [119]. However, it has not been yet validated whether the existing curricular are also suitable and effective for younger and more entry-level students. Since new standards and principles for teaching younger groups of students at entry or introductory level with little computing experiences still need to be established, this work will focus on CS education for novice students at the secondary level to provide new insights into how introductory programming needs to be developed and implemented for younger students.

1.2 Current Learning Resources for Introductory Programming

Developing better tools to enhance learning of students with little exposure to computer programming has been an active research area in the field of computer science education. Most effort has been made in developing tools that better engage students in programming activities. Interacting with computer systems using programming languages requires understanding on computer architecture (e.g., memory space, file systems), programming language syntax, control flow, programming paradigm, and domain-specific high level logics and concepts. For students who do not have enough background or experiences, therefore, it is critical to develop the educational resources such as software tools that are dedicated to familiarizing them with these aspects.

"Scratch [99]" is another well-known example of educational tools for introductory programming. Using Scratch, learners can easily create projects by assembling color-coded blocks of programming elements; they can combine block-like operators to generate animated motion of objects, handle user-input events, or manipulate the control flow of the program. Moreover, it is also being used for teenagers or even younger children, because creation of interesting programs is relatively easy and can be online-based for social interaction and collaborations among the users. It supports collaborative projects through its built-in feature of "remixing" by which a user can add their own products onto the top of others' projects. According to the large-scale analysis on the data from Scratch projects, sharing projects for collaborations can serve as a pathway to learning and improving computational thinking [35]. Thus, Scratch provides an example of how collaborative learning can be effectively used for education of introductory programming.

On the other hand, there is an alternative view that more general purpose languages such as Java, Python, or JavaScript should be directly introduced in programming education, rather than starting with Logo or Scratch [78]. It has been suggested that exposing novice students to such full-featured programming languages is a better way to motivate them [53] and more practically applicable [93], compared to teaching education-purpose programming languages first. Several attempts have been made to help novice users learning the general-purpose programming languages with different aspects of programming. For example, “DrJava [4]” provides interactive REPL (read-eval-print loop) interface to provide an easier debugging environment for novice users. “Python Tutor [52]” provides step-by-step visualization of internal operations so that novice users can better understand what’s happening in heap, stack, and console area in the execution of the code. “Pythy [40]” is a cloud-based application platform that abstracts away installation of development tools, file management, and searching for example codes. These examples demonstrate that much effort has been made to to reduce novice users’ learning curves in programming practice.

Taken together, the previous research work in the field mainly focused on identifying the appropriate programming languages and educational tools for teaching introductory programming and syntax: Logo simplifies drawing mechanism based on units of geometry; Scratch abstracts away coding process and supports the block-based visual programming via its drag-and-drop UI; and Python Tutor provides visual cues for internal operations on memory structure in executing user’s code.

1.3 How Can CS Education Support Collaborative Learning?

The overarching goal of my dissertation work is to foster collaborative programming environment for novice students in CS. The literature review in the previous sections (1.1 and 1.2) suggests that the previous CS curricular and research work on introductory programming education have heavily focused on teaching the syntax of programming languages to individual users in an isolated, non-collaborative learning environment. Although mastering programming skills is an important aspect of CS education, there are also other areas to be considered as the fundamentals of CS education rather than just being comfortable with writing code: Being a CS major is much more than learning how to code [55]. Therefore, my dissertation work first addresses the important aspect that has been widely overlooked in the previous literature and CS teaching curricula: effective approaches to implementing collaborative programming environments to the introductory education for beginning students.

K-12 CS framework [44] expects that students who received CS education at the secondary level should be able to utilize their collaboration tools to build complex software by effectively working together with other students. In reality, however, collaborative learning environment is not easy to be implemented in CS classrooms and fostering it is also an under-explored research area. Although collaborative learning method such as pair programming, is known to be a very effective strategy, most of these approaches are not closely linked to or built in the CS curriculum, but is considered to be rather supplementary. Furthermore, even the most widely used platforms for younger students' collaborative programming, such as Scratch, do not effectively support collaborations that involve direct social interactions between users. In addition, its features are not sufficient to support the complex and dynamic aspects of collaborative programming projects in the real world settings. Therefore, my dissertation work focuses on how to better implement collaborative programming in the CS curriculum, so that the skill sets acquired in the classroom can be extended and applied to real-world industrial settings.

1.4 Outline of the Dissertation

In Chapter 2 of my dissertation, we will first review the literature on social development theory by Vygotsky [127] in the field of developmental psychology. The social developmental theory will provide the foundation of the roles of social interactions in social, cognitive development and learning. This theory also has significant implications for the framework on interactive, multi-user computer systems, especially for the purpose of learning. The social development theory will also provide insights into general principles for designing learning activities, defining the roles and the types of interactions in learning of beginning students, and facilitating their meaning construction [70]. To establish effective methods for the social interaction in learning, we will then review the existing literature on computer-supported collaborative learning (CSCL) [65]. The CSCL literature will provide us with better understanding of what types of interactive process using computer software would lead to successful collaboration in learning contexts and how to support this. Chapter 2 then will end by discussing and high-level design principles that will potentially help designing effective and novel approaches to implementing a collaborative environment to CS classrooms.

Chapter 3 will describe the preliminary study that explores software-structured collaborative learning. It will illustrate the empirical work that assessed the feasibility of a specific software design to structure users' behavior on it and their task performance. we will first describe the preliminary experiment on collaborative learning performed using a software tool, named "MOOCchat". MOOCchat is an educational software tool to facilitate chat-based peer learning processes in MOOCs (massive open online courses) settings. It creates ad-hoc discussion groups for users who are available in the online waitlist in each of the periodical quiz sessions. It scaffolds interactions between users in these discussion groups while they solve the quiz individually and collaboratively. The findings of the study suggest that 1) the software design with the sequential phases effectively guided the participants to the intended behavior for collaborative learning and 2) the collaborative approach to problem solving through the text chat in small groups significantly improves the performance of the users.

Chapter 4 will explore how students learn to collaborate with other students in the real-world learning environment, rather than a simulated context described in Chapter 3. We will describe a qualitative study on the instructional approaches to teaching novice users web programming in real-world classroom. This study conducts interviews with teachers to characterize the potential challenges and problems in implementing collaborative learning environments to the lab sessions at secondary level. We will discuss these potential challenges by relating to the theoretical frameworks on cognitive load and cognitive apprenticeship. To summarize the findings, the qualitative data analyses from the interviews revealed three major challenges. First, collaboration is surprisingly unstructured in lab activities, suggesting the need of better structures to be implemented to collaboration. Second, there seems to be a mismatch between the types of knowledge students need for the task and the instructional strategies. Finally, teachers encounter difficulties in successfully convincing their inexperienced students to appreciate the importance of the collaborative process and its consequences.

Chapter 5 will discuss how these challenges can be addressed when implementing collaborative learning. It will propose a novel approach to implementing collaborative programming with lesson plans and a supporting software tool for the lesson plans. It will further illustrate more specific requirements, design considerations, and implementation strategies for the proposed approach. Based on the proposed approach, it will also introduce a software tool named "GLIDE (Git-Learning Integrated Development Environment)". GLIDE provides users with useful resources and guidance to a simplified version of standard Git workflow. Chapter 5 also describes the design-based research (DBR) that explores the deployment of the proposed approach to teaching collaborative programming in a real-world classroom. Specifically, this observational study looked into how this approach affects the process and the outcomes of the class projects by novice students. The findings from the observation and focus group sessions showed that utilizing an appropriate teaching method will allow high school seniors to understand the concepts on the structured collaboration process and perform group projects by following the proposed workflow, even when they only had basic literacy in web client programming (HTML, CSS, and JavaScript) and just a little experience in team projects.

Chapter 6 will quantitatively evaluate the performance of the students after the classroom intervention with GLIDE. In this field experiment, we examined the effects of the intervention on students' experiential and attitudinal outcomes. The findings suggested that better understanding of the basic concepts in structured collaboration and scaffolding educational tools facilitated relevant communication between the students and improved qualitative aspects of learning, including better self-efficacy, class engagement, psychological ownership of the group projects, and perceived equity in project coordination.

Finally, Chapter 7 will conclude by discussing the implications of this work including contributions to the field, limitations, and future directions of this study.

CHAPTER 2

LITERATURE REVIEW

This chapter reflects on the existing literature that provides theoretical basis for this dissertation work builds on. The literature review starts with social development theory as the theoretical foundation to understand how learning occurs in this dissertation. And then, it discusses prior works on computer-supported collaborative learning (CSCL) in CS, as a more concrete approach to learning through social interaction with the guidance from computer software.

2.1 Social Development Theory

Social development theory [126, 127] is a learning theory that views learning as internal developmental processes where knowledge and abilities are internalized through social interactions. Vygotsky [126] used the concept of developmental level, which indicates functional, psychological, and mental maturity of a student in explaining where and how learning occurs. The actual developmental level (ADL) means the conceptual domain that a student has already matured into, so that she has developed the ability to independently solve problems corresponding to this domain. On the other hand, the potential developmental level (PDL) means the higher level domain where she can solve problems with guidance from "the more knowledgeable other (MKO; for example, a teacher, parents, or peer collaborators)". This learning theory argues learning that aims the PDL facilitates a student's development, which is otherwise ineffective. This makes a point of the concept of zone of proximal development (ZPD) as where learning occurs, indicating the distance between ADL and PDL. Thus, in order to facilitate a student's internal development, social development theory suggests that the MKO should provide the appropriate assistance for tasks or knowledge in the student's ZPD through interaction with her.

Social development theory roots in the significance of tools and sign systems through which social interaction occurs. Vygotsky [125] believed that language, as the representative tool and also sign system for social interaction, is very important in learning for two reasons. One reason is that, when a student intellectually develops in ZPD with guidance from MKO, language is the main means for the information exchange during the social interaction. This idea upholds the general theme of this learning theory that learning is a socio-cultural process because language itself has been formed under the cultural development of a society over the course of human history. Hence, learning through social interaction using the language, which is socially accepted and agreed on, includes the broader and more abstract sense of interaction with the society as well as the direct interaction with the MKO in learning context [127]. Another reason is that a student actively uses her own language in the process of internalization to adapt the information that came from the social interaction. The active process of internalization, rather than passive copy, of knowledge that came from social origins is the central element of Vygotskian learning theory [46]. A student uses her personal interpretation in the form of language when this process takes place. Thus, this learning theory focuses on the tools and sign systems, such as language, in the core part of the learning process. This idea has become the pivotal foundation of the constructivist learning theory that learning is an active, interactive, and contextual process, rather than a unidirectional and didactic process, through which all knowledge is constructed from the learner's experience and interpretation [15].

Existing research took Vygotskian approach to better understand novice students' learning in CS. Gillespie and Beisser [48] explored how young students age of seven learns Logo programming language. The teachers who partnered with the research team had the pedagogical basis of Piagetian learning theory, which weighs more on the children's biological development and their own construction of knowledge [90, 133]. However, the young children's learning process turned out to be better aligned with Vygotskian theory, according to the observation that they learn through social interaction with MKOs including adults and more competent peers. Deepening this discussion, Whalley and Kasto [131] identified the patterns of ZPD of novice students in learning Java programming. They observed the developmental levels where each participant can solve problems either independently or with a teacher's situational help while taking programming tests with three different complexity levels. Their findings, in consistence with the theory, suggested the importance of use of language both for interaction and internalization as a useful indicator of a student's learning. Participants who performed well in the tests were able to interact with the teacher using their own language, whereas those who didn't had difficulty in tracing and explaining their solutions. Another approach was taken by Hundhausen [57] where he reported positive learning outcomes when he had the students present animated explanation of algorithms they implemented in an undergraduate algorithms course. This work supports the Vygotskian argument that students learn better when they're given a chance to represent the knowledge they constructed in their own language and interact with others using their own representation. Moreover, this study pointed that algorithm visualization the students constructed was effective in mediating the student interaction, which expands the interpretation of tools and sign systems in Vygotskian learning theory into computer-mediated interaction. Despite the influence of the learning theory on CS educators and these examples of existing research, it is not often explicitly discussed in CS education, compared to the vast amount of literature on it in science and mathematics education [16]. This research tries to continue the discussion relevant to the socio-cultural learning theory into CS education for novice students.

From the perspective on how learning occurs provided by social development theory, this theory guides my work to the clear design of lesson plans that implements a novel approach to learning collaborative programming in CS. First, this theory provides a framework for planning and designing the learning context using the introduced concepts of ADL, PDL, and ZPD. This will directly help this dissertation work specifying the lessons with what developmental level the potential target students have generally matured into (ADL) and what functions and knowledge are learnable, falling in their ZPD, considering their PDL. Second, the gravity placed on ZPD in this theory advises that teachers should design "guided participation" from students in the learning process. Learning functions and knowledge in PDL, by definition, requires help from MKO, so guided participation denotes the way to facilitate learning through interactive and collaborative ways. My dissertation work focuses on the design that guides students to learn practical functions and underlying concepts in collaborative programming in interactive ways as suggested. Lastly, this learning theory informs this work about what roles tools and sign systems play in learning. This theory indicates that language is used to support both social interaction with others and internalization on a student's own. Inspired by Hundhausen's work [57], my dissertation continues using the extended meaning of tools and sign systems as language to represent ideas and knowledge, which provides a guideline for the design of the tools this research proposes.

2.2 Computer-Supported Collaborative Learning

Collaborative learning is an educational approach where students work together in solving problems or completing tasks [47]. It stems from the idea that learning occurs in a social context when the learners actively participate in the social interaction [109], which implies a thread of connection with Vygotsky's social development theory. As one of the instructional methods to support collaborative learning, this section reviews the existing literature on computer-supported collaborative learning (CSCL) that involves software tool as the mediator of collaboration. Also, as existing literature says the effectiveness of collaborative learning varies according to the specific learning settings [39], this section focuses on CSCL approaches dedicated to support learning in computer programming in CS.

Existing research made diverse attempts to take advantage of CSCL approach in CS. One substantial class of them is based on the idea of sharing the artifacts to build. For example, Fields et al. [43] studied collaborative learning for high school students who built animated music video using Scratch. They designed the programming task to involve the multi-level structure of collaborative support; each student was guided to come up with the individual segment of music and also build on top of multiple layers of contribution from group members, mentors, teachers, and unknown Scratch users online. Their findings claim that this specific way of "nested" design of the task revealed potential inconsistency of program state among the different levels of collaboration, which brought about the students' learning on the concept of initialization. This type of collaborative programming is supported through project-sharing mechanisms, such as "remixing" [35, 36, 77]. On the other hand, Urai et al. [120] explored various tool supports for pair programming practice in distant settings. Pair programming is a typical example of collaborative programming where two programmers work together with the dedicated roles of driver (who writes code) and navigator (who reviews code written by the driver). Although the genuine form of collocated pair programming doesn't necessarily take CSCL approach, there have been several attempts to support distant communication in pair programming in the shared project through a synchronized code editor [18, 38, 84, 118]. To sum up, CSCL approaches in this category provided asynchronized sharing features for existing projects that help students to build on top of others' work or synchronized co-work mechanisms on the shared project.

Another significant chunk of works in CSCL in CS is to support students to communicate with each other sharing their representation of concepts to learn. HabiPro [124], for instance, is a software tool designed for students to collaboratively solve programming-related problems. The tasks included identifying errors, reordering code segments, predicting the return values, and completing a partial program. It has a text chat tool where a group of students can discuss the problem to solve. The authors argued that this approach doesn't directly teach how to program, but gives novice students a chance to reflect on their code and observe others thoughts, which is one of the critical aspects of collaborative learning. Similarly, CAROUSEL [58] takes constructivist learning approach to computer algorithms by allowing students to build multimedia representation in explaining algorithms. The students were also allowed to discuss and evaluate the algorithm representations that other students generated won the tool. The results of the quantitative evaluation on this approach show that the students engaged in creating their own explanations and improved their understanding through the representation process. In sum, several CSCL approaches in this category primarily focused on providing verbal or graphical communication channels for social interaction among the students, which consequently help them learning concepts and practice of programming [29, 71, 95, 102, 108].

2.3 Foundations for Theory-Driven Approach to Collaborative Programming Education

The review of the literature on social development theory and related works in CSCL provides the theoretical framework of my dissertation. As a theory-driven approach to learning in collaborative programming, this work designs the collaborative learning environment and empirically tests the design approach, where the learning context roots in Vygotsky's learning theory and the concrete method follows the principles in CSCL.

First, informed by the learning theory, my dissertation considers what sign systems and tools to provide in order to facilitate learning. Existing works in CSCL have proposed a lot of novel tools supporting social interaction among students during their learning process. Even though this looks well aligned with Vygotsky's theory that learning occurs from social interaction, there has been little consideration on the sign systems and tools that mediate the students and social context, but not only the learning context, as the socio-cultural products. The sign systems and tools in which the students communicate should help them (indirectly) interact with the broader sense of social context, be part of the social context that has generated the tools, and internalize the socio-culturally constructed knowledge gained through interaction for their own mental development. This suggests that a potential approach to learning in collaborative programming should actively embed norms, terminology, affordances of software, and widely accepted processes to collaborate in the learning context. Israel et al. [60] also pointed that collaboration models being used in actual classrooms, such as teacher-facilitated model and peer tutoring model, don't fully reflect on the realistic model of collaborative programming, implying the discontinuity between the learning context and the social context. Therefore, this theory-driven approach aims to design the learning context that reflects the socio-cultural aspects of collaborative programming in real-world practice.

Second, grounded in the concepts of ZPD (where learning occurs) and guided participation (the method to facilitate learning in ZPD), this work will take scaffolding approach to facilitate novice students' learning in PDL. Scaffolding is one of the most widely accepted ways to interpret guided participation [86, 101]. Wood et al. [132] first coined the term "scaffolding" to describe how teachers and parents can help learning in guided ways. They argued that the effective teaching requires two concrete ideas: 1) the conceptual model on the task with how it may be completed and 2) the conceptual model on the performance characteristics of the student. Combining these two, a teacher can design the instruction where she reduce degrees of freedom, redirect students, and marking critical features, or demonstrate her own task performance. In response to this, successful examples of CSCL approaches in CS included well-structured task designs along with the software tools supporting the tasks. Thus, this work tries to follow scaffolding approach to facilitate novice students' learning in ZPD.

Lastly, this work tries to extend the existing literature on CSCL by embedding computer-supported collaboration in lab activities of programming. In overall CSCL literature across the disciplines, mediated communication in verbal language or other digital representation through software tools is one of the most common ways to support learning in selected lessons. Literature on CSCL in CS also rendered the foundation of collaborative learning in similar ways. However, CS courses, especially programming courses, might have more diverse opportunity to take CSCL approach since their coursework materials, assignments, and lab activities are already built on computer systems by its nature. Besides, the historical perspective on CS curricula has been calling for better support for lab activities in CS courses. Thus, this work will suggest a mediating software tool for computer-supported collaborative programming as a lab activity.

CHAPTER 3

STUDY1: BRINGING STUDENT INTERACTION INTO ONLINE LEARNING

3.1 Motivation

The foundations for theory-driven approach to collaborative programming education (section 2.3) call for needs of dedicated software tools that mediate student interaction and guide their task process in learning context. Along with the theoretical foundation as a starting point for the rest of the dissertation work, it is also important to understand the practical aspects of collaborative learning structured by software and task designs that implement the theme of computer-mediated collaborative learning. This chapter describes the first empirical study that explored such software-structured collaborative learning. The purpose of this study is to test out a relatively simple and rough approach to support collaboration in learning in CS and learn how students react to the proposed design.

Online education is often identified as a learning environment with limited students interaction because students are neither collocated, nor are they progressing through course materials on the same schedule. That is, they are isolated in terms of both space and time, which makes online education a domain calling to engage in challenge to support their social interaction for learning. Existing CSCL literature has shown several examples of text chat-enabled online workspaces where students perform individual tasks with extra communication channels [102, 124], but there has been little attention paid to collaborative learning that involves online social interaction as an integral part of the tasks. This study tests out a novel design of structured collaboration tasks with students interaction in online CS education.

This study is inspired by the literature on structured peer learning [83, 111]. Peer learning in physical classrooms structures students activities in which they have discussion in small groups to come up with solutions to problem-solving tasks. The significant pedagogical benefits of peer learning as a practical teaching method, including improved critical thinking skills, retention of learning gain, interest in subject matter, have been well documented by literature in several disciplines [26, 37, 76, 110]. Hence, this study implements a simple version of structured peer learning for online CS courses.

The idea of introduction of peer learning into an online education platform gets well aligned with the theoretical background of this dissertation work. One rationale is that peer learning, as a practical teaching method, encourages students interactions in the learning context, so that the interactions facilitate inter-psychological development [127] stated in Vygotskian learning theory. Another way peer learning in online education roots in the theory is that the students are required to actively process information for their own meaning construction while participating in discussion, rather than passively accept it. This helps the internalization process [46] whose important role in learning has been pointed out in social development theory.

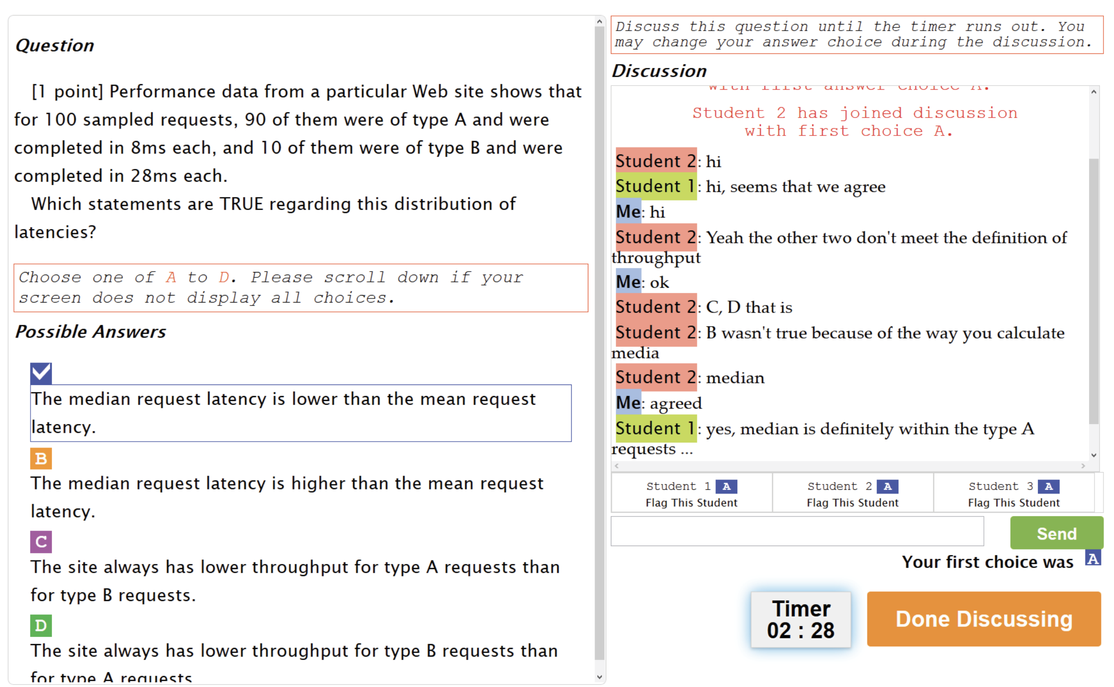
3.2 Methodology

3.2.1 Participants

61 students taking the online course of CS.169.2x Software as a Service on edX participated in this study. They were recruited through the course announcement and their participation was voluntary. The announcement stated there won't be monetary compensation or advantages in grading upon the participation, but encouraged their participation for the benefit in their own learning through collaborative learning tasks (practice quiz sessions, explained below) designed toward improved learning outcomes and experience. The six-week course was intended for students with an undergraduate CS major level of expertise. Given the MOOCs (massive open online courses) settings, it was hard to collect accurate and detailed demographic information of the participants.

3.2.2 Design of Software and Tasks

We built a software tool named MOOCchat. MOOCchat is a tool to run collaborative quiz sessions designed to be integrated with MOOCs web platforms. The main features of it included real-time grouping of users who are online, assigning quiz materials to each group, sharing the answers from individuals within the group, and allowing them to discuss using a timed chat tool (Figure 3.1). To bring students together into synchronous groups in online education settings, approaches from team formation in multi-player games and real-time crowdsourcing [75] was adopted; the "bus terminal model" was implemented so that a new quiz session begins at regular time intervals if there are users waiting online. MOOCchat is designed to form as many triads as possible, dyads if less than three users are remaining among the waiting users. In cases where there is only one user remaining online, MOOCchat was supposed to serve a non-collaborative practice quiz session for the user. MOOCchat was a web-based application built with Node.js.



Question

[1 point] Performance data from a particular Web site shows that for 100 sampled requests, 90 of them were of type A and were completed in 8ms each, and 10 of them were of type B and were completed in 28ms each.

Which statements are TRUE regarding this distribution of latencies?

Choose one of A to D. Please scroll down if your screen does not display all choices.

Possible Answers

☑ The median request latency is lower than the mean request latency.

B The median request latency is higher than the mean request latency.

C The site always has lower throughput for type A requests than for type B requests.

D The site always has lower throughput for type B requests than for type A requests

Discuss this question until the timer runs out. You may change your answer choice during the discussion.

Discussion

with first answer choice A.

Student 2 has joined discussion with first choice A.

Student 2: hi
Student 1: hi, seems that we agree
Me: hi
Student 2: Yeah the other two don't meet the definition of throughput
Me: ok
Student 2: C, D that is
Student 2: B wasn't true because of the way you calculate media
Student 2: median
Me: agreed
Student 1: yes, median is definitely within the type A requests ...

Your first choice was A

Figure 3.1: A screenshot of MOOCchat (the initial version)

MOOCchat was deployed in the Fall 2013 offering of edX's CS.169.2x Software as a Service. The intervention provided students with an online collaborative practice (ungraded) quiz session that they could opt to take using MOOCchat integrated with the course website. MOOCchat structured the online quiz task in three stages; 1) MOOCchat organizes students into small groups, 2) it gives them a chance to individually answer a question, and then, 3) it displays everyone's responses and places them in the group's private chat room so they can discuss the answer. These three stages of collaboration were guided by a server-controlled timer counting down to keep the group of students in sync. When the timer runs out, the students were able to have the final chance to change and submit individual answers. Finally, they were then shown the correct answer with an explanation. They then moved on to the next question to repeat the three-staged quiz session (Figure 3.2). The questions were composed by the instructor of the online course to help them review and better understand the course materials.

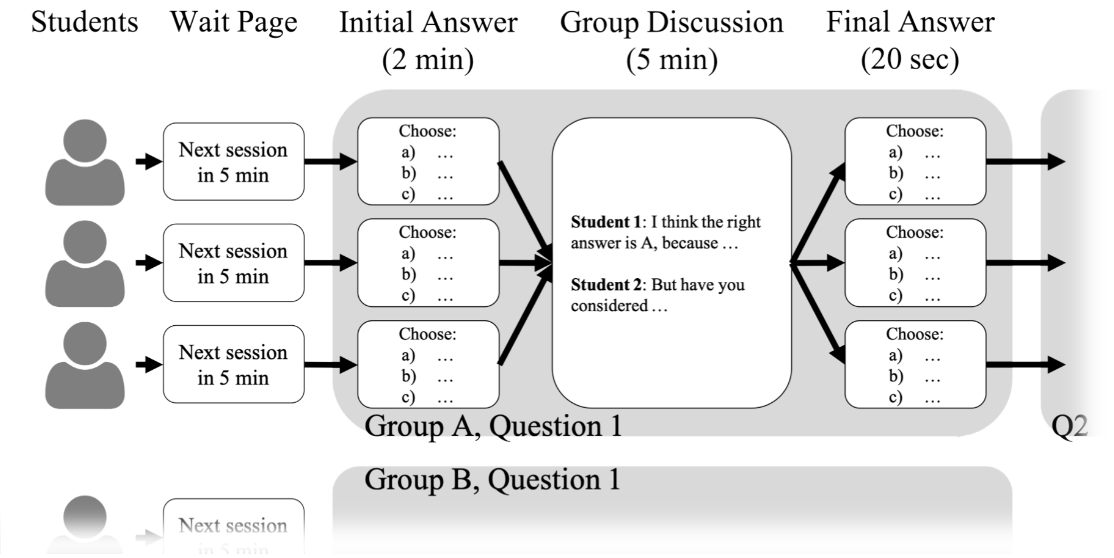


Figure 3.2: Illustration of structured task: collaborative quiz sessions

3.2.3 Data Collection

To understand students' initial experience and thoughts on the software-structured collaborative task, we gave them brief survey questions at the end of the quiz session. This survey was to get qualitative understanding on participants' reaction to the tool and the method, rather than statistical analysis for a confirmatory study or hypothesis testing.

3.3 Results

61 students took the practice quiz and completed the survey, which began every hour. Of these, only 16 were successfully placed in a discussion group of 2 or 3 participants; six of them were placed in groups of 3, and ten were in groups of 2 (More groups were not formed because not enough students arrived at the same time for the same session; a large MOOC with thousands of students could have a higher grouping success rate). The focus of this report is on the experiences of those students who did have a discussion with others using the chat tool to draw implications for collaborative learning.

Table 3.1 shows the post-quiz survey results, indicating overall positive responses to this online intervention. In response to "Other students helped me learn during the discussion," one student wrote "Yes, by having to explain my answers to the other students it forced me to think more deeply about the question," which was one of the central tenets behind peer learning. In terms of group size, 6 out of 10 of those in dyads indicated they wanted more people in the discussion whereas everyone placed in triads indicated this was the right size for the discussion, reflecting results from the peer learning literature that dyads do not lend themselves to good discussions.

One of the survey questions was an open-ended one (not shown in Table 3.1) asking "Do you have any feedback about your experience using this discussion tool? What worked well and what can be improved?" The students expressed general satisfaction. one student commented that "It was my first time using

Table 3.1: Survey questionnaire and responses from MOOCchat participants

Survey Question	Agree / Strongly Agree	Neutral	Disagree / Strongly Disagree
Discussion was helpful for final choice	11	2	3
I was able to help others learn	9	4	3
Other students helped me learn	9	4	3
I liked discussing questions in a small group and would like to do so again	14	1	1

this. I think that overall it is a great tool. We were able to have some brief discussion and it probably is the closest thing that we can get to being the same activity that is in the course." Another wrote "That is very interesting, useful and fun. Cool." A third wrote "It's really cool, and make learning more interactive!!!" These comments suggest that students in an online course are quite positive about this approach. Also, small coordinated group chats may successfully lead to better learning and retention as has been found widely in the peer learning literature.

3.4 Discussion

We have developed MOOCchat and an instructional method to form synchronous discussion groups in MOOCs using the tool. Also, the software tool and the collaborative task design were tested out in an actual online course. It was the first step in understanding how software can guide students collaboration in an online education environment, but much work remains to be done.

The first takeaway in terms of the software design is that the staged model works very well in structuring students behavior in using the tool. It seems that the students perceived the linear staged model straightforward, judging from no attrition observed out of 61 students who began the task. None of the open-ended feedback comments in the survey had negative report on the difficulty in using the tool or following the collaborative task. More encouragingly, the staged model worked well in guiding the cognitive tasks (e.g., reading passages and figuring out the answers) as well as the behavioral tasks (e.g., clicking button to proceed and typing texts for discussion). This could be useful design implication for future works.

As the survey feedback shows, MOOCchat successfully facilitated students interaction around the task. The task design wise, it seems that the separation of individual task from collaborative part (discussion) and deliberately using the outcomes of individual task (answers) in the collaborative task (sharing the group members' answers) contributed to the feasibility of performing the task, judging from the responses to the survey question "Discussion was helpful for final choice". For further investigation, we conducted another iteration of MOOCchat in a crowdsourcing platform to verify the quantifiable effectiveness of this approach. In that experiment, participants who were grouped for the collaborative tasks marked higher rates of correct answers in the quiz, compared to solo participants; among the 169 responses provided by solo participants, 84 (50%) were correct, where among the 269 responses from crowd workers who participated in a discussion group (dyads or triads), 169 (63%) were correct. This difference turned out to be statistically significant (Fisher's two-tailed exact test, p ¡ 0.01). This supports that MOOCchat approach successfully facilitated the students interaction, which played a key role in performing the tasks

[31].

This study also had several limitations, which opens the possibility for the next step going forward. The online learning environment renders a lot of potential for novel approaches to learning with lower cost and higher accessibility, however, it also rendered serious defects as a research testbed. For example, the synchronous online collaboration is heavily affected by disperse attendance at any given time because online learning environment heavily relies on an individual's self-directed or self-paced learning, which is on the opposite of structured learning. It might be possible to strictly schedule the participation to alleviate this issue, but this scenario less reflects on the real-world learning context as a natural field test. Consequently, the MOOCchat experiment left almost three quarters of the overall voluntary participants ungrouped for the collaborative task. This can't be the ideal setting for the research trying to develop and test collaborative approaches to learning with students in the actual learning context.

We argued that MOOCchat approach effectively facilitated interaction. However, given that the students could answer multiple choice questions and collaborate to identify and correct errors in those answers through text chat, there is no evidence that the interaction has brought about learning; a possible alternative explanation is that the students had a chance to give a second look into their answers when others in the discussion group express different opinions, or even more simply, they could have corrected the answers just following others' opinions. MOOCchat approach could be justified for its own purpose that the review and self-evaluation through peer discussion on the coursework materials were much needed to help the specific target students learn. However, the findings are hardly sufficient to expand to social learning in introductory programming practice due to the use of the ad-hoc grouping and chat tool. Possible reasons include the simplicity of the task (answering multiple choice questions), the way of collaboration (a free-form text chat for a limited period of time), the level of engagement (self-directed online learning not being graded). This calls for further investigation in students tasks taking longer term, with higher complexity, which better engages them in and in better structured ways.

Another limitation, considering the general goal of this dissertation aiming to provide a social learning method for collaborative programming, was that the collaborative quiz task we designed didn't provide the authentic experience in programming or yield potentials to iterate in that direction. Given the course materials that addressed programming issues and practice (Ruby on Rails), giving Ruby-related questions for collaborative quiz sessions would have been as close as MOOCchat approach can get to programming education through social interaction. Even with those questions, it would have been more of another replication of syntax-oriented learning in programming with extra communication channels among peers, which doesn't fulfill this dissertation's vision that introductory programming education should be designed toward collaboration.

3.5 Summary

We conducted an experimental study to explore the practical feasibility of software-structured collaboration and to learn about how students behave on and react to it. This study designed MOOCchat approach that facilitates students interaction around collaborative quiz sessions and deployed it to an online CS course. This method organized students into ad-hoc small groups to discuss their answers and rationale for the answers, to employ peer learning as the instructional method where students exchange their ideas in learning from others. The post-task survey results said that students were very positive about both the experience and the effectiveness of this approach. We concluded that MOOCchat successfully brought much desirable students interaction in an innately isolated online learning environment, but there wasn't enough evidence that actual learning occurred through their interaction. This calls for more careful consideration on research domain and programming tasks that more directly connect to learning for future works and also better understanding on actual learning environment.

CHAPTER 4

STUDY 2: "WORK TOGETHER ON WHAT AND HOW?": POTENTIALS AND CHALLENGES IN TEACHING COLLABORATIVE WEB PROGRAMMING

4.1 Motivation

The preliminary study led us in a new direction; we felt the necessity of exploring classroom environments and how students are guided to learn in introductory programming courses. One reason is that online learning environment that relies solely on each individual student's self-directed study may not allow us to introduce a novel approach with a focus on guided participation or "well-structuredness" and observe the students behavior; due to unstructured progress of individual students, decontextualized observation, low turnout rates, and extreme selection bias of the participants. Also, to make intervention into a physical classroom environment, we need to better understand how students are currently learning. Thus, this chapter explores how students are guided to learning in an engaging CS topic for novice students.

Web development has become one of effective topics for engaging novice students in CS. Teachers and CS education researchers have made diverse attempts to teach web-based technologies in the CS curriculum [3, 69, 72, 78, 80, 96, 115]. For example, Mercuri et al. [80] used web programming languages for introductory programming, finding that website-building is highly engaging for students. A similar argument was made by Stepp et al. [115] in his quantified evidence on students' gradual curve of perceived difficulty and constantly high enjoyment across sub-topics of a web development course. Furthermore, such enjoyment resulted in an increase in student's motivation for taking CS advanced courses in their investigation. These examples also illustrate the importance of lab activities in these courses, where students learn through practices and projects, and the effectiveness of engaging novice students in these rich web-development activities.

To better support novice users' lab activities in web programming courses at the secondary level, this chapter identifies rooms to introduce collaboration. We investigate how teachers structure these kinds of lab activities and what element of the activities we can reconstruct to involve students collaboration, based on their reported success (or lack thereof) of these approaches from their experience. Specifically, we asked high school and middle school teachers about what tasks they do with their students and what instructional approaches they adopt for these tasks, in their introductory web development courses. Understanding how these courses are taught in middle or high school context will give us an opportunity to broaden our understanding on challenges and potentials in instruction for novice students.

4.2 Related Work

Existing research has explored several challenges in introducing collaboration in learning environments. Janssen et al. [61] viewed novice users' immaturity, as an individual factor, in coordinating group members as one of the most significant challenges. Their observation showed that working collaboratively may impose additional cognitive costs for novice students. This is because collaboration involves metacognitive activities that regulate their task performance, such as planning the whole task, monitoring progress, and evaluating the task [41] in which novice users are generally underdeveloped. Another line of work argued that there is social factors including competition in education system and relationship issues [12], which might demotivate students collaboration. Furthermore, cultural factors were reported including teachers' potential reluctance to bring collaboration in their classrooms where they consider it as an operating cost [100].

To extend the discussion, we explore instructional factors that make it difficult to introduce collaboration in web programming courses. As our research shows, web programming involves diverse tasks, including analysis, design, and deployment related activities, but little is known about how to use collaboration as a teaching strategy to support these relevant tasks in lab activities in those courses. We seek to address this gap by creating a taxonomy of these tasks, along with what instructional approaches have been used to introduce then, and how students were guided to work together by our sample of teachers.

4.3 Methodology

4.3.1 Participants

We recruited 11 (five female and six male) voluntary participants for data collection through snowball sampling. They were sampled with common criteria of teaching experience in web development courses and 3 or more years of teaching experience in CS. They had experience of teaching different age groups of students in different settings. The participants with pseudonyms and the characteristics of their teaching context are shown in Table 4.1.

4.3.2 Data Collection

We conducted semi-structured individual interviews. The purpose of these interviews was to understand how participants design and deliver lab activities in teaching web development courses. The interviews followed a prepared protocol to get key information common to all participants, but as a semi-structured interview, it also allowed us to actively ask unscripted questions to be informed about further details or other relevant issues. The questions covered topics of curriculum, technology supports, and the class activity process. Example questions for the topics are shown in Table 4.2. Seven participants joined the interview in-person, while four did remotely using video conferencing software over the Internet. The interviews were approximately 40 minutes in length.

Seven participants agreed to share any teaching materials (e.g., curriculum, exercises, etc.) they used along with examples of project outcomes of their students. These supplemental materials assisted us to better understand the interview data by providing information on what's been done in student activities.

4.3.3 Data Analysis

The interview recordings were fully transcribed and analyzed. The analysis started with the first step of the grounded theory approach, which is open coding [116]. Open coding, as an independent analytical method, has also been

Table 4.1: Interview participants for exploratory study

No.	Pseudonym (Gender)	Teaching Environment	Experience in Years	Course Subject	Supplemental Materials
1	Ann (female)	Public high school	5	Web development	N/A
2	Bob (male)	Private high school	15	Software engineering	Curriculum, student exercises
3	Carter (male)	Public high school with software engineering concentration	4	Software engineering	Student projects
4	Dana (female)	Public high school	7	Mathematics	N/A
5	Eric (male)	Public high school	3	Web development	N/A
6	Floyd (male)	Public high school with software engineering concentration	14	Web development	Curriculum, student projects
7	Gina (female)	High school level after-school program, community college	3	Web development	Curriculum, student projects
8	Helen (female)	Public middle school, middle school level club activities	4	Infographics	Student projects
9	Irene (female)	Public high school	Did not reveal	Web design	Student projects
10	Jay (male)	High school level after-school program	6	Web development	N/A
11	Ken (male)	Private high school	12	Web development	N/A

Table 4.2: Examples of interview questions per topic for teachers

Topic	Example Question
Curriculum	"What kind of curriculum do you use for teaching web development?" "What are the high-level learning goals for the curriculum?"
Class Activities	"What kind of activities do you have your students conduct?" "Could you explain the students' task process in that activity?"
Technology Support	"What tools do you have your students use?" "What materials or starter code do you provide?"

known to be useful in extracting concepts and ideas from the data, even without the complete process of axial coding and selective coding of grounded theory approach [82, 89]. During open coding on the interview data, references on noteworthy phenomena in the data are iteratively coded with emergent names that concisely describe the phenomena. The phenomena of interest for this research included tasks that students performed in lab activities and strategies used by teachers to support students in those tasks. After extraction of the concepts through open coding, we tried to identify patterns of strategies.

4.4 Theoretical Background

Cognitive load theory [85] and cognitive apprenticeship [32] were used to generate systematic perspectives on the emerging codes and their categories. Use of these two theories was theoretically justified because they have common premise that learning can be explained by task complexity and cognitive load, which have been indicated as two factors contributing to difficulties in learning [92]. We used these two theories in our coding scheme to theoretically understand how learning is supported across the various instructional designs applied by teachers in our sample.

4.4.1 Cognitive Load Theory

From a cognitive science perspectives, learning occurs when new information being processed through working memory becomes part of existing patterns of memory structure (schema) in long-term memory [85]. This process is generally called schema acquisition. Cognitive load theory [64] states that learning outcomes of a learner can be maximized when the new information gives her the adequate level of cognitive load, which is high enough to fully activate schema acquisition and also low enough for schema acquisition to happen. From this point of view, controlling this variable through instructional design is the key to effective learning [64]. Studies have shown that this can impact the effectiveness of learning [24].

Research in cognitive load theory has explained, at a more concrete level, how instructional designs help a learner's knowledge acquisition. Caspersen and Bennedsen [22] provide an overview of the four effects of instructional design; worked examples effect, example completion effect, variability effect, and expertise-reversal and guidance-fading effect. Here, we focus on the first three effects as they are most directly intended for novice learners. Table 4.3 shows how these instructional designs were used in our coding scheme.

Table 4.3: Instructional designs according to cognitive load theory

Instructional Design	Definition	Example of use in web development context
Worked Example [98]	Learning from teacher's demonstration of example of given working solution	Students analyzing and studying the code of existing websites
Example Completion [122]	Learning through modifying and extending given working solution	Students completing or customizing template HTML and CSS code that is tailored for specific lessons
Variability [88]	Learning through comparing and contrasting several variations of worked example	Students comparing similar HTML elements with different parameters of box model in CSS

4.4.2 Cognitive Apprenticeship

The theory of cognitive apprenticeship says that skills or knowledge acquisition happens when the learner (novice) intentionally uses it through being able to see the teacher (expert)'s processes of the task [32, 91]. This learning theory draws attention to tacit steps in performing a task, which novices often fail to observe because those steps are not very visible in complex tasks [32].

According to Collins et al. [32, 33], cognitive apprenticeship has four aspects of learning, or expertise transfer; modelling, scaffolding, fading, and coaching. Modeling aspect means that knowledge transfer takes place when the expert directly shows the novice how to perform tasks to set the "model case" of task processes that the novice can learn from. Scaffolding means partially carried-out task by the expert that is given to the novice to support their work. Fading is gradual removal of the expert's support, so that the novice can become eventually more responsible for the task. In coaching, the expert supervises the

Table 4.4: Instructional designs according to cognitive apprenticeship

Instructional Design	**Definition**	**Example of use in web development context**
Modeling [33]	Learning from direct observation of how experts perform a task	Teacher demonstrating how to upload website resources to an FTP (File Transfer Protocol) server
Scaffolding [33]	Learning from performing a task where the task was partially carried-out by an expert to reduce the complexity of it	Teacher giving her students a semi-automated shell script for setting up the web server as scaffolding
Coaching [33]	Learning from performing a task where an expert supervises the process and gives meta-level help, such as feedback, hints, encouragement, and additional challenges	Teacher giving her students free-form projects with few requirements and constraints with rather high level feedback

overall process of the task by giving meta-level help, such as feedback, hints, encouragement, and additional challenges.

Collins et al. [32] suggested that there exists interplay among those four aspects of cognitive apprenticeship in knowledge transfer, which implies that these concepts are not mutually exclusive. However, from an instructional design perspective, a teacher might strategically and selectively emphasize specific aspects of them in designing her instruction. Table 4.4 summarizes how these instructional designs were used in our coding scheme.

4.5 Results

4.5.1 Task Types

From the participants' responses, six general task types emerged; analysis, design, implementation, code sharing, deployment, and review. The definition of these tasks and including examples from the data for each task type is shown in Table 4.5.

4.5.2 Instructional Designs Interacting with Task Types

Six of the instructional designs outlined in the theoretical framework were identified to be employed by the participants to support students' tasks for lab activities; worked example, example completion, and variability in cognitive load theory and modelling, scaffolding, and coaching in cognitive apprenticeship. Based on their self-reported success by the participants, we constructed the instructional design and web programming task matrix (Table 4.6). As qualitative research, this work calls readers for analytical generalization of characteristics of the emergent findings, instead of statistical generalization of frequency of references or count of features visualized [82].

In Table 4.6, the two theoretical frameworks that we introduced are vertically arranged. Cognitive load theory sees learning as acquiring schema and processing information to fit in the schema as conceptual knowledge [79]. On the other hand, cognitive apprenticeship regards learning as knowing how to do things, cognitively or physically, as procedural knowledge [79]. The task types

Table 4.5: Task types in web programming courses identified

Task Type	Definition	Example Reference
Analysis	Tasks directly related to studying existing artifacts as given materials to understand the technical details or identify problems	"They have already done X-Ray Goggles. They have looked at a bunch of websites. And we have talked about design. What's good and what's bad."
Design	Tasks directly related to studying given materials to understand the technical details or identify problems	"I give them grid paper. They make a paper wireframe sort of thing. When they do that, I let them first try out. Draw whatever you want."
Implementation	Tasks directly related to building digital artifacts with program code	"They would use a cloud editor to change parameters and queries to have Twitter, Instagram, and SoundCloud like web interfaces."
Code Sharing	Tasks directly related to make one's program code available to others or to use other's code	"They would send code by email or use Google Docs, which is not super compatible. That was a source of frustration. Or, copying from the screen and typing."
Deployment	Tasks directly related to make digital artifacts available and accessible to others on the web	"Students would really love to have a hosted website. For one semester, I did, like I set up a classroom server and I had the students, like taught the advanced students, how to host the sites."
Review	Tasks directly related to evaluate digital artifacts and building process after completion of building	"I also let them have reflections at the end, to make sure that they were involved with the process and usually if there has been gross imbalance of work."

Table 4.6: Reported success of instructional designs per task type

Base Theory	**Instructional Design**	**Task Type**					
		Analysis	Design	Implementation	Code Sharing	Deployment	Review
Cognitive Load Theory	Worked Example	🟦					
	Example Completion			🟦			
	Variability	🟦					
Cognitive Apprenticeship	Modeling				🟥	🟥	
	Scaffolding			🟪		🟥	
	Coaching		🟥	🟪	🟪		🟦

🟦 Instructional design reported as successful

🟥 Instructional design reported as unsuccessful

🟪 Instructional design reported as mixed success

□ Unidentified instructional designs

are horizontally arranged in general task order of development cycle (although it's an iterative process, not a linear sequence). Given that, the diagonal pattern of the shaded cells, from top-left to bottom-right, suggests that the participants wanted their students to acquire schema to understand concepts in the earlier stage of lab activities and pick up skills to build artifacts later.

The sparseness of the matrix suggests conformity in selection of instructional approaches per task type across the participants; white cells in a vertical column might suggest the participants had a tendency to use common approaches to the specific task. In contrast, a column with more cells colored might imply that the task allowed the participants to try more diverse instructional design approaches. This holds true when we consider the characteristics of each task and the nature of knowledge the task is involved with. This pattern again resonates with the theoretical types of knowledge aforementioned [79]. Another notable pattern in the matrix is that implementation was the only task type where both theoretical approaches were identified; in particular, example completion was used to support students to learn programming language syntax.

The participants largely used cognitive load theory approaches to support the analysis task, while using cognitive apprenticeship approaches to support more diverse task types to build artifacts. Another notable feature in the visual patterns is that implementation was the only task type where both theoretical approaches were identified; example completion was used to support students to learn programming language syntax.

4.5.3 Collaborative Approaches Interacting with Task Types

In addition to the instructional designs guided by the theoretical framework, collaboration between students emerged as a key aspect of the participants' instructional strategy in the observation. The participants reported that they tried three types of collaborative approaches; peer instruction, cooperation, and discussion. The definitions and examples in web programming context of the three emergent collaborative approaches are shown in Table 4.7.

We also tried to assess the relative success of collaborative approaches per each task to find potential opportunities for better support. These collaborative approaches, as instructional designs did, emerged interacting with task types, also showing the general conformity of selection in the collaborative approaches and task type matrix (Table 4.8). To get a systematic perspective on the results and identify patterns, one intuitive way to arrange the three collaborative approaches is by the levels of intrinsic structuredness of the method, where peer instruction imposes the most structured way of collaboration and

Table 4.7: Collaborative approaches identified

Collaborative Approaches	Definition	Example of use in web development context
Peer Instruction [34]	Engaging students in understanding the core concepts presented, and then, explaining them to their fellow students	“I tried pairing them up like strong students with weaker students, so they help them along in coding.”
Cooperation [61]	Dividing the labor required among fellow students to solve problems or complete tasks together	“I say to them you can work up to a group of 6.”
Discussion [67]	Encouraging verbal interactions among fellow students to help engaging in a lesson and learn academic content by exchanging ideas	“I ask a question, ‘how does it work?’ They have a discussion about it. I give them the notes about, on a basic level what are functions and events in JavaScript.”

discussion allows free-form exchange of information for collaboration. Given that, the inverted U-shape pattern across the development cycle suggests the tendency that teachers take more structured approaches to hands-on stages of the development process and allow more discretion for analytic steps, in terms of the way of collaboration (Table 4.8). One notable feature of the visual patterns was that design task did not often or successfully involve structured collaborative approaches. One possible explanation could be that novice students find the design task hard because it usually includes high-level thinking and sets the foundation for the remaining tasks, while it also should allow high degrees of freedom for them to be creative at the same time. Nonetheless, this seems a crucial missed opportunity for introducing more and better structured collaborative activities.

From the patterns of instructional designs and collaborative approaches identified along with their reported success, we examined the potential to in-

Table 4.8: Reported success of collaborative approaches per task type

Collaborative Approach	Task Type					
	Analysis	Design	Implementation	Code Sharing	Deployment	Review
Peer Instruction			🟦	🟦	🟥	
Cooperation			🟪			
Discussion	🟦	🟥				🟦

🟦 Collaborative approach reported as successful

🟥 Collaborative approach reported as unsuccessful

🟪 Collaborative approach reported as mixed success

□ Unidentified collaborative approach for the task

troduce collaborative approaches that are supported by instructional designs per task. A description of these results including relevant interview excerpts are presented below.

4.5.4 How to Support Discussion for Analysis and Review Tasks

The participants confidently stated their successful teaching experiences in supporting analysis and review tasks. They had their students discuss in performing those tasks while providing instructional designs of worked examples, variability, and coaching as the additional support. This piece of findings tips at the possibility of synergic effects from the adequate combination of collaborative learning and instructional designs. For analysis task, it seems that worked examples design facilitated students communication through setting up the common ground [56, 28] and variability design help students contrasting different consequences of the dynamically changing common ground in the discussion.

Below, Ann talks about students discussion with instructional design support of worked examples design for analysis task and Carter explains how his students discussion works with variability design for analysis task.

> Ann: "We look at evaluating websites, then we actually look at HTML and CSS [of them]. Some of the kids get to Javascript but that's like an extension. Then the websites are graded on both web development and the content. I printed out their favorite websites like the ESPN home page. A lot of boys really like sports. I found like a fashion site because a lot of girls like fashion, not to stereotype (laughs). That is what they asked for. They drew boxes around the different parts and we labelled the tags. We did things like that, marking things up. Even though they really don't need to know how to build it from scratch, we started there, so that they understood the fundamentals."

> Carter: "They basically looked at the code, and answered the questions about, 'How do you think this piece works?', or 'What do you think it does?' I got about 30 kids who responded which is pretty good for my class They copy and paste the function that alerts and changes the background color. Copy and paste the line which changes the background color to green. They just hunt through and are like 'Oh! This must be it, background color, green!' Then, I give them a different[, but similar] code. 'How does it work?"'

For review task, coaching approach of cognitive apprenticeship was found to be successful. Review task seems to work well with discussion and coaching design because it requires the ability to evaluate an object with relevant criteria, which is one of the representative metacognitive skills [41]. As Irene reported below, she uses the term 'scaffold' to explain the strategic assistance she provided her students, but it actually looks closer to coaching because she gives meta-level assistance, rather than scaffolding in cognitive apprenticeship.

> Irene: "They sit in their table groups, and they have worksheets of a particular format on how to give feedback. And then they look at everyone else in the table group and fill it out just for them. And then they hand it back. If there is a group of four, everyone will get three forms with the feedback. And then I have to do a few lessons on how to incorporate feedback. They are also not good at giving feedback, so they are not always getting meaningful feedback. So, I do try to scaffold so that they get some meaningful feedback."

4.5.5 What Works Well or Not in Supporting Implementation Task

The participants reported generally mixed success on implementation task. Certain collaborative approaches and instructional designs are theoretically known to be effective, which led the participants to make several attempts at implementing those approaches, but caveats and barriers were still identified in practice.

First, Example completion design and peer instruction were identified to have successfully supported this task. We couldn't find the direct interaction between the two methods in our dataset, but peer instruction in implementation task with example completion design looks a promising combination. Bob explained how he guided his students to build on top of a given working solution (a layout of an app) by modifying and amplifying it, which is example completion design. Eric stated that peer instruction was generally a successful collaborative approach in his classroom.

> Bob: "You have fresh blank Ruby file in Cloud9. We put in the Sinatra framework in there. But I don't want to do [teach how to use the module imported] that in every class. So, we have standard gems already installed and we have a basic layout. We have a get request and post request set up."

> Eric: "I tried pairing them up like strong students with weaker students, so they help them along. I tried giving strong students their own projects that they have to teach [the partners] how to do."

Second, scaffolding design reported mixed success in supporting implementation task. We made comparisons between the attempts that ended up with mixed success to find out any potential factor that generates different consequences. Gina came up with two contrasting results in supporting implementation task with scaffolding design, even in the same course.

> Gina: "We had these starter code where we tried to figure out the OAuth and have the code for that just like 'done', and then they would download the zip file and upload it to their server. Each of them had a little folder on a GoDaddy [a web server]. And then, they

> would use a cloud editor to change parameters and queries to have Twitter, Instagram, and SoundCloud, like web interfaces [APIs]. We kind of styled towards them, they started changing out the CSS, colors, and stuff like that. That was probably one of the successful ones."

> Gina: "One thing that no one's got was data binding [of JavaScript template engine]. We used Handlebars templates, where you could see what the data parameter you received was. And there would be like div tags [for the output of the data parameter]. They kind of understood the data somehow went in there, but the dynamic generation, like dynamic rendering of HTML, I don't think really ever took hold. I bet it could take hold, but I just haven't figured it out how to."

In the two anecdotes, Gina provided partially carried-out code of which her students were able to build on top. Even though OAuth and RESTful APIs (Application Programming Interface through Representational State Transfer) are much harder topics than the dynamic rendering of HTML, she reported her scaffolding approach was successful for the former rather than the latter. We suppose that this mixed results stemmed from the mismatch between the learning goal, the type of knowledge to teach, and the instructional design. What makes scaffolding effective is hiding the complicated part of the task from students, while still letting them learn in performing the other part of it, especially, taking advantage from the partially carried-out part [33]. If the learning goal is to understand how a system works, as in the second anecdote of Gina, scaffolding doesn't seem to help. In addition, the type of knowledge she wanted to teach looks conceptual rather than procedural in this case. Thus, it is advisable that teachers should clarify the learning goals and the types of knowledge to teach when using scaffolding design to support implementation task.

Third, coaching design also reported mixed success. We could also make comparison between cases reporting on coaching design with different consequences. Below, Ann illustrated the successful meta-level support she provided in a form of checklist. Floyd, on the other hand, described unsuccessful example of using coaching design focusing on feedback support.

> Ann: "It was just [building] a static website. Then they had the checklist requirements. There was an HTML checklist and CSS checklist. Things like 'it has to include a video', 'it has to include an ordered list', things like that. So, they used the checklist and they are evaluated on the rubric. So, like 'how readable is the website', 'how well designed it', etc."

> Floyd: "As I was introducing CSS to them, I think that the syntax itself was an order of complexity that was beyond the HTML. They never get the hang of the idea of getting braces and semicolons to define rules for different tags. The process would be they would type something in, and then they would stare at it, they didn't know what to do next, they would raise their hand, and then the teacher come over and say 'Oh, you need a semicolon there' and 'Okay'. They would type something in, and then it wouldn't work, they would raise their hand, and teacher would come over and say 'Oh, you need a brace over there.' That was kind of tedious."

Both responses described meta-level support, however, the latter wasn't a successful coaching design. We suppose that this issue stemmed from the centrality of the information given as meta-level help; coaching is giving meta-level support, but not giving central information as meta-level support. In Floyd's example, CSS syntax is rather crucial for the task. Thus, it is advisable that teachers should prioritize better structured instructional design in giving information that is central to the task. Provided peer instruction was identified as successful collaborative approach for implementation task, another possible suggestion is that coaching design that involves peer instruction where fellow students exchange meta-level support, such as hints or ongoing feedback.

Finally, Carter commented on students' social relationships and the school's contextual factors as potential causes for the mixed success of cooperation in supporting implementation task.

> Carter: "I don't really do group work because of the nature of my school. There is attendance issues, the kids who just drop-off, there are kids who no one wants to work with because they have emotional or behavioral issues. Forcing that group dynamics upon them can lead to disaster of classroom engagement and culture."

4.5.6 What Makes It Hard to Teach Design, Code Sharing, and Deployment Tasks

The participants stated that they have tried several different approaches without discovering satisfying solutions in design, code sharing, and deployment tasks, even though a few approaches were found to be more effective than others. These tasks seem to have more room for improvement as well as significant potential for collaborative learning and instructional designs. In this section, we look into the challenges identified in supporting design, code sharing, and development tasks.

Design task seriously lacks structured support, in terms of both collaborative approaches and instructional designs. Floyd shared with us his experience where he wanted his students to freely design on paper before they started coding, but his coaching approach to give them flexibility was not found to be effective for novice students.

> Floyd: "So, anytime I have them draw something on the paper, then, put it on the screen, it's a little bit of an argument. They really just want to put something very quickly on the paper to appease me, and then start playing on the computer to try to create it. I see the value in design, but it's hard to get the design in the way something get out to graphically on a piece of paper. That's been hard to motivate them to see the value of that."

Code sharing and deployment tasks require further technology supports. Code sharing, by its nature, has two or more peers working together, which implies further potentials for collaborative approaches. Peer instruction was identified as a rare successful approach for code sharing where sharer and sharee learn from each other. Ann showed that code sharing with coaching approach through peer instruction was successful, given that the tool her students used supported code sharing feature embedded. Also, Carter gave an example showing an unsuccessful modelling approach to code sharing, where the tool he gave his students was too hard to use for them. Similarly, deployment requires further tool supports in Gina's anecdote since novice students didn't fully understand web architecture in having the artifacts they built hosted on a web server.

> Ann: "Well, Students would really love to have a hosted website. For one semester, I did, like I set up a classroom server and I had the students, like taught the advanced students how to host the sites. They did it for everyone else. Everyone linked their site to the master site and I taught these students how to... I did not do that after that semester. Just because it took so much of the class time. And the majority of the students weren't able to really get anything out of it."

> Carter: "This year, I did it [code sharing] using version control. So, our kids had to clone from GitHub and had to push back. It just drives them nuts. I have recorded videos on how to do it, I have a poster in every classroom. I have helper students who know how to do it, but still, you know, the syntax [command] is a huge struggle. I would say with the HTML they did pretty decent."

> Gina: "We used for a while this tool called 'ShiftEdit'. It basically allow you to upload files and edit them. I had to first buy server space on GoDaddy, web client. And then, I made everyone register for ShiftEdit account, and then I gave them credential information for GoDaddy to connect their ShiftEdit to that, for every single student. This step takes forever. They had to just follow this tutorial."

4.6 Discussion

We formulated three propositions on instructional approaches to better support collaborative lab activities in web development courses based on the findings presented above.

4.6.1 Align Knowledge Types, Instructional Designs, and Task Types.

We derived the proposition that teachers should be careful about aligning. One rationale is that this will help avoiding the mismatch between types of knowledge to teach, collaboration approaches, and instructional designs to use. The unsuccessful anecdote of Gina in section 4.5.5 showed examples of the mismatch. In this case, she took scaffolding approach in teaching conceptual knowledge of how JavaScript template engine works. Even though her students were able to make the module work, Gina's excerpt implies that it wasn't enough for the intended teaching goal. This resonates with literature on the relationship between conceptual knowledge and procedural knowledge. Glaser [50] explains conceptual knowledge helps effective use of procedural knowledge. This proposition suggests expanding the literature to embrace that instructional designs should match the types of knowledge to teach, according to the learning goals and the characteristics of tasks.

4.6.2 Have Concrete Plans for a Certain Task and How Instructional Designs or Collaboration Would Help.

The unsuccessful approaches called for more structured designs. The participants responded they have tried several methods, but they were largely unsuccessful. One of the common challenges observed was that their guidance was extremely unstructured. Several collaborative approaches, for example, were found in a form that "do this task together" or "discuss this within your group". Stahl et al. [113] stated that collaborative learning must be designed in detail, so that the design is coupled with the analysis of the meaning constructed during the task. This proposition suggests task-by-task analysis that clarifies what learning goal you have, what the target students would find difficult during the task, and how collaborative approaches or instructional designs would support them to perform the task.

4.6.3 Teach and Support End-to-End Collaborative Workflows.

Although collaboration was identified to be a common approach with various forms in the cases we observed, there was no end-to-end collaborative workflow that carried through each of the steps. This kind of collaboration, with well-defined roles, responsibilities, and processes for division and integration of work, is a core aspect of most professional software engineering contexts. Floyd stated that he wanted to teach collaborative programming in a better structured and pipelined way as a whole procedure, while emphasizing how difficult this is in practice.

> Floyd: "I think having someone be able to break the problem down into functional sections, then distribute it to have the sub-problem solved, and put it back together, it's just something they struggle at that level. One project that we did at the freshman level, it's in Python. They were trying to make the alphabets and I said to them, 'you can work up to a group of 6'. Everyone who can, they divide up the letters. I found that one student do A through F, and then G through J or whatever, and then, the lesson actually ended up being an example of not to do. It ended up with one person does A through F, and then the person who's supposed to pick up G got confused. The one with A through F is done, and then they think it's all just gonna fit together. And then, nobody was actually able to make somebody else's code work on theirs because they all put it in the different sizes and they can't put it on the screen. The one major collaborative project that I did with my freshman last year was more of introduction to what the problems are. I tried to explain it to them beforehand like what's going to go wrong, but they're really not interested in to learn and see it go wrong, so it's an interesting lesson, which I enjoyed doing, but it more accentuates the problems than solves them."

Several different kinds of tools, including bug trackers, task assignment tools, and version control systems, are widely used to support the pipelined collaboration process in professional software engineering contexts. However, teaching young students how to use them is another big challenge [59]. By designing appropriate tools that guide students through end-to-end collaborative workflow, this sort of challenge that Floyd was reporting might be alleviated.

Also, instruction on the collaboration workflow using those tools might reduce the gap between collaborative processes in education environment and the field of software engineering.

This research has a few limitations. First, our research design did not take factors outside of the instructional designs employed into account. In particular, there may have been external factors that could help or limit strategic approaches to learning. These potential external factors might include school policy on flexibility of curriculum, management costs, availability of tools and instructional materials, and students' prior skill levels. Another limitation is this research used self-reported measures of effectiveness of instruction by teachers, not an objective assessment of the instruction or the evaluation on students' learning achievement.

4.7 Summary

Based on a semi-structured interview study with middle and high school web development teachers, we explored how these teachers employed instructional design approaches and collaborative approaches to support students' lab activities in web programming, and the relative success of these approaches. From these findings, we derived and discussed some potential strategies to improve instructions.

CHAPTER 5

STUDY 3: OBSERVATION OF HOW NOVICE STUDENTS LEARN A SCAFFOLDED COLLABORATIVE WORKFLOW

5.1 Motivation

In the previous chapter, the qualitative study identified the challenges in supporting students collaboration in web programming courses in secondary level CS classrooms: 1) we discovered misalignment between the knowledge types to teach, the instruction designs to employ, and the task types students are assigned with; 2) the plans for teaching tasks didn't have concrete plans for collaboration; and 3) there was a lack of support for end-to-end workflow dedicated for collaboration in web programming activities. As the discussion section of the previous chapter suggested, these challenges commonly call for more concrete plans on how to have students collaborate on what tasks in each stage of web programming activities. Thus, we propose a CSCL approach including lesson plans and a supporting software tool to alleviate these issues. Also, we deploy the proposed lesson plans and the software in the actual education settings to observe how the novel approach turns out in real world contexts and how students adjust to the proposed approach to collaborative web programming.

There are several different ways available to introduce collaborative programming in CS classrooms, but even the most widely used platforms for younger students' collaborative programming, such as Scratch, do not effectively support the complex and dynamic aspects of collaborative programming projects observed the real world settings; specifically, they don't always reflect or aren't applicable to the common tasks in the development cycle that we identified. One widely accepted tool for collaborative programming is using version control systems (VCS). A VCS is a software tool to manage changes of program code over time [8]. Using VCS has become a crucial skill to manage one's own codebase, share your contributions one others, integrate multiple set changes, and to make this traceable and reversible in a collaborative situation. Despite their significance, teaching students how to use VCS has been left out of the standard CS curriculum up until very recently [8, 7, 6]. As a result, students are not introduced to how collaborative software projects are organized in real-world settings, and do not appreciate the significance of collaboration. Moreover, the absence of tools and protocols for collaborative project management may result in reduced work efficiency and unfair contribution for student projects, increased managerial costs for teachers to guide the students' work process or handle conflicts between students, and delayed learning in advanced courses that could build on top of effective collaboration. Existing approaches to collaborative projects without the structured guidance of VCS, accordingly, might be missing the elements of professional programming practice, such as working with teams and solving collaborative challenges by sharing and merging code through VCS. Therefore, this chapter focuses on designing and deploying curriculum and a supporting software to help students at pre-collegiate levels learn how to effectively work with each other using standard collaborative programming techniques.

This chapter first reviews related work on how collaborative workflows have been introduced in CS education. To expand the discussion to younger age groups of students and project-based web programming courses, we propose lesson plans where students go through genuine process of collaborative workflow using VCS. We introduce GLIDE, a software tool built around the proposed lesson plans, on which students can collaboratively build websites according to the feature branch workflow of Git. And then, we report on the deployment of the lesson plans and GLIDE in an actual CS classroom at a high school. In sum, we provide insights into how to better teach collaborative programming through our proposed approach and an empirical study as an evaluation of the proposed approach.

5.2 Related Work

Several research projects have used VCS as a courseware to manage, distribute, and submit learning materials and students deliverables [19, 68]. For instance, Lawrance et al. [68] suggested a structured model of using Git in courses for CS majors and also CS1 courses for non-CS engineering majors. This included introducing Git command line tools and configurations. In the resulting study, some of the CS majors who participated reported that they were able to get jobs based on learning Git.

Other projects have introduced VCS as an explicit learning goal, and encouraged students to use it for collaborative development within the course (and hopefully even after the course) [66, 54]. For example, Laadan et al. [66] employed Git in an operating systems course for undergraduate and graduate students. They introduced the benefits of VCS as courseware, and the use of version control in software development processes. The students participated in group projects about kernel programming using Git, allowing the instructor to review the code contributed in group projects. Students enjoyed communicating with the instructor through comments, suggesting the benefit of collaboration using VCS in CS courses.

Our research builds on, but differentiates from, these prior works in several ways. First, prior works commonly pointed out that the learning curve in using VCS, and the fear thereof, is one of the biggest challenges to introducing it into the curriculum [59, 54, 97]. This issue is exacerbated for non-CS majors who take CS courses and younger students because they are unlikely to have sufficient background: for example, using the command shell prompt, understanding the terminology of VCS, and having a conceptual model of a commit tree. Second, prior software tools were not designed to teach the process of collaborative workflow with proper scaffolding; focusing on learning sequences of commands by rote memorization [20, 21, 68, 97], providing a graphical user interface (GUI) to automate the entire process [123, 17, 54], or providing platforms for unstructured collaboration, such as "remixing" [35]. Third, most prior work focuses on the post-secondary level. By introducing collaboration tools and processes early in high school, we can encourage pedagogical approaches based on social learning and collaboration that can support students in further CS learning.

5.3 Proposed Approach and Research Questions

5.3.1 Educational Context

We chose to design lesson units to be taught in an introductory web programming course offered at the secondary level. The target audience is novice students without sufficient background in systematic development process in groups, but having a basic understanding in syntax of HTML, CSS, and JavaScript. The basic web architecture, which usually requires background in those three computer languages, makes good settings by nature encouraging collaboration with contributors from each area. Also, existing research has shown that web programming can be engaging and enjoyable to novice students in introductory courses [80]. Considering the topic and the skill levels, we targeted 12th graders in a high school who are taking a web programming course. We distributed a call for participation in this research to over 50 teachers at high schools in New York, NY. One teacher who has been teaching a web programming course at a high school in Queens, NY for eleven years agreed to participate. We planned an intervention including the lesson plans while consulting the instructor of the course to get informed about the course organization and students' technical background. She gave us feedback on the feasibility of each lesson (decision on whether her students would be able to follow each lesson or not), the time required for practice, and general tips for class session management.

5.3.2 Lesson Plans

In designing our lesson plans and supporting tools, and materials, we focused on the following two learning goals. First, students should be able to collaboratively write programs using a Git workflow that supports structured collaboration. Second, not only being able to write programs, students should be able to understand the process of how collaborative software development should occur, rather than leaving the process management onto the scaffolding tools that abstract away the procedure or getting trained in a set of instructions that they can mindlessly follow. These two goals target learning that better reflects collaborative programming process in professional contexts. Using Git, we chose to teach the feature branch workflow. The feature branch workflow refers to a pattern of software development wherein a group of developers use a shared repository for a project and incrementally update it by pushing individual contributions using VCS [7]. In the standard Git workflow, each developer clones the original project repository (often called the upstream or the origin) into their own local branch, which they can use as an active workspace. When she is ready to contribute her changes (commits), she can push them to the origin, where a VCS handles the interaction between the two separate codebases. Upon the approval by the repository owner, the branch with updates is merged into the master (default) branch of the repository. During this process, collaboration occurs by reviewing and merging multiple branches from all the group members. These explicit steps of the workflow as a collaborative model contribute to a more maintainable codebase and more explainable codebase history. Figure 5.1 depicts the conceptual step-by-step procedure of the feature branch workflow.

The key concept we want to teach about the feature branch workflow is how a group of developers can individually make progress on their own branches and then integrate their code by merging their branches into master branch. This doesn't only have practical importance in collaborative programming, but also conceptual significance that involves computational thinking, such as abstraction of a state of code (a snapshot) and a tree-like representation of changes of dynamic entities. In addition, The feature branch workflow is a good approach for novices for three reasons. First, as a basic variation of central workflow [6], it is one of the simplest workflows for novices to understand that is

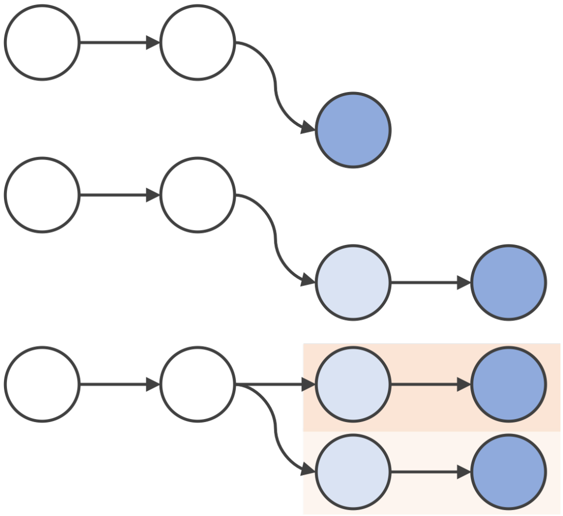


Figure 5.1: A conceptual diagram of the feature branch workflow. The diagram illustrates a new branch checked out (the curved arrow), two new commits (the colored circles) made on the branch, and the branch merged into master branch.)

known to work well with small sized groups. Second, this workflow is adequate for novices because it tries to keep the representative branch of the project repository from broken code [7]. Lastly, the separation among the original project repository on a remote server and a contributor's workspace keeps the project from potential errors compared to direct modification off the original project repository.

Two important lessons to start with that will help the students perform group projects following the proposed workflow is the basic terminology used in Git workflow and the abstract model of Git operations (not Git commands) in each step of the workflow; for example, what a repository means, what cloning a repository means, and what happens when you clone a repository. Teaching these two prerequisites is also relevant to the learning goal that students should have deep enough understanding to engage in communication on the collaboration process using technical terms. Thus, this proposed approach exposes

Table 5.1: Proposed Lesson Plans

No.	Lesson	Duration	Description	Materials
1	a. The motivation: why we need Git? b. The concept of repository	60 mins	a. Students share experience in manually managing versions using different filenames. b. Students understand the difference between file folders and repositories.	
2	a. My GitHub account b. My first GitHub repository c. Repository is a folder being traced	30 mins	a. Students create their own GitHub account b. Students create their own remote repository on GitHub c. Students edit README.md file and get familiar with the file system browser on GitHub	Web browser and email address
3	a. Individual contribution b. Remote repo and local repo c. Master branch and feature branch d. Commits as save points	60 mins	a. Students understand the abstract model of feature branch workflow. b. Students understand the concepts and terminology covered. c. Students discuss how Git works with peers in their own words.	
4	a. Cloning on GLIDE b. Branching on GLIDE c. Coding on GLIDE	30 mins	a. Group leaders invite the members to the GitHub repository. b. Students clone the team repository on GLIDE. c. Students create their own branches in the cloned repository on GLIDE. d. Students create HTML files on the branch on GLIDE.	Web browser, GitHub account, and GLIDE
5	a. Putting them all together b. Pushing feature branch c. Merging branches	60 mins	a. Students understand push, pull request, and merge. b. Students understand the concepts and terminology covered. c. Students discuss how those Git operations work with peers in their own words.	
6	a. Pushing and Merging b. Deploying the website	30 mins	a. Students push the branch on GLIDE. b. Students resolve merge conflict on GitHub. c. Students configures GitHub pages to serve their repository as a website.	Web browser, GitHub account, and GLIDE

them to the terminology and the corresponding operations of Git for each step. Table 5.1 summarizes the lesson plans reflecting the learning goals.

5.3.3 GLIDE

We built a software tool named GLIDE (Git-Learning Integrated Development Environment) [73] to guide novice students using scaffolded user interface based on the feature branch workflow. It is built to have all the features required to support the lab activities in the lesson plans (Table 5.1) and a student group project to build a static website as a classroom activity after the lessons.

GLIDE supports students in visualizing the workflow as they proceed through each step of the standard workflow. Figure 5.1 illustrates the interactive navigation bar on the top position of GLIDE user interface. It is based on a simple, static, and iterative 5-step cycle: clone, branch or checkout, code and test, commit and push, and make a pull request. Each step (except for code and test, which are not Git-related operations) represents one abstract Git operation covered in the proposed lesson plans (Table 5.1).

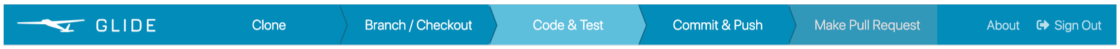


Figure 5.2: Interactive navigation bar of GLIDE illustrating the scaffolded feature branch workflow

GLIDE structures a group of students' web programming process after brainstorming, coordination, and design phases like this (Figure 5.2):

- The group leader creates a remote repository on a Git hosting service provider, such as GitHub.
- The leader invites all the group members in the project repository to grant them the access to the shared resources in the repository.
- Each student opens GLIDE on one's own computer and clone the project repository.
- Each student creates a branch to build a feature on; he writes HTML, CSS, and/or JavaScript code and test it on the live preview window.
- Each student makes commits to confirm the changes and pushes the branch.

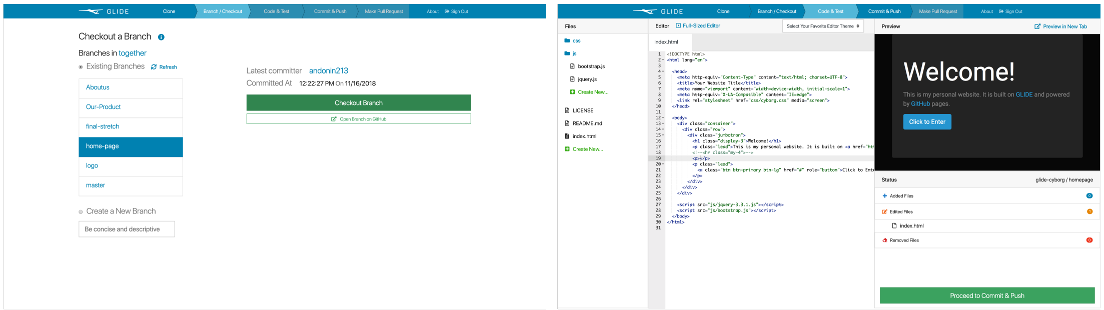
- Each student repeats this cycle until the work is complete and make a pull request when it's ready.
- The group members get together to review the code and merge the branch into master branch on GitHub, upon the decision by the group or the group leader.

This represents the scaffolded feature branch workflow where the novice students can collaboratively perform website-building project even without learning Unix shell command language or Git command line tools, such as "git checkout -b my-branch" or "git push origin my-branch", by using GLIDE.

We also wanted to make it easy for students to deploy their websites using GitHub Pages [49]. We have found that young people are excited to share their work online and to show their friends and family. However, students and teachers find it hard to perform and support deployment given the complicated set of requirements (buying a domain, setting DNS parameters, finding a hosting provider, uploading content, etc.). GitHub Pages, are a built-in feature of the Git hosting provider, where the static website resources can be served directly from the remote Git repository. Another GitHub-related feature GLIDE employs is making a pull request to notify the repository owner (the group leader) that a branch has been pushed and ready to be merged. This feature took classroom context into account where students might perform group projects as homework after having to wrap up their work within the limited lab session. In such distributed settings, this pull request feature works as a means for communication.

Figure 5.3 shows a couple of screenshots of GLIDE user interface. GLIDE was built with Django Rest Framework (Python 3), React (JavaScript), and

GitHub API to interact with the remote repositories of a user.



Latest committer andonin213

Committed At 12:22:27 PM On 11/16/2018

```
<!DOCTYPE html>
<html lang="en">

  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Your Website Title</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <link rel="stylesheet" href="css/cyborg.css" media="screen">
  </head>

  <body>
    <div class="container">
      <div class="row">
        <div class="jumbotron">
          <h1 class="display-3">Welcome!</h1>
          <p class="lead">This is my personal website. It is built on <a href="ht
          <!--<hr class="my-4">-->
          <p></p>
          <p class="lead">
            <a class="btn btn-primary btn-lg" href="#" role="button">Click to Ent
          </p>
        </div>
      </div>
    </div>

    <script src="js/jquery-3.3.1.js"></script>
    <script src="js/bootstrap.js"></script>
  </body>
</html>
```

This is my personal website. It is built on GLIDE and powered by GitHub pages.

Figure 5.3: Screenshots of GLIDE user interface

As the initial experience of integrating introductory web programming courses at pre-collegiate levels with a collaborative Git workflow, we need to investigate how this approach turns out in an actual education settings. Thus, we formulated research questions on the consequences and effects that the GLIDE approach potentially brings about.

5.3.4 Research Questions

The proposed curriculum using GLIDE implements a novel approach to teaching collaborative web programming with the guidance of structured process based on Git workflow. Despite the significance of collaboration in CS education context, little is known about the feasibility and the consequences of introducing collaborative workflow into secondary level CS classrooms. The results from the interview study in the previous chapter showed that teachers acknowledge the value of teaching standard process of collaboration and are motivated to teach it, however, the existing curriculum has less focus on learning how to collaborate. Furthermore, there were no well-known software tools to support teaching and learning such a collaborative process. Along with these unfulfilled needs, collaboration using Git workflow has been left out of the secondary level CS curriculum, while creating the perception that Git is too hard to teach or learn at beginner levels. As we argue that our novel approach using GLIDE make the implicit steps of collaborative workflow more visible and comprehensible, we need to investigate the learning gains that the novice students take from the proposed approach. Thus, this study raises the first research question of *what traceable or observable learning gains novice students can take from the proposed approach to collaborative web programming*.

The intended contribution of this dissertation work in the bigger picture renders the second research question on how the proposed approach contribute to fostering a collaborative environment, in addition to learning in collaboration skills. An active research topic in CS education research is how to deal with the issues of unequal contribution in collaboration [14, 104, 105]. This implies that collaborative learning in CS has attracted much attention from researchers and educators, but the fairness of collaboration has been a practical barrier working against the implementation of collaborative environment. Aligned with our general expectation that the proposed approach will guide students to fair and explicit coordination as feature branch workflow organizes the group work, the second research question asks *how the proposed approach alleviates the unequal contribution problem in collaboration, and in turn, contribute to fostering a collaborative environment for CS education*.

5.4 Methodology

To answer the research questions, we conducted design-based research (DBR). DBR in learning science refers to a research approach that designs a context to generate new theories, artifacts, and educational practices that potentially serve and affect learning and teaching in natural settings [11, 10]. Cobb et al. [30] characterized this research methodology that it entails both "engineering" certain forms of learning and studying those forms of learning within the context defined and designed to support them. This justifies the use of DBR in this research where we designed lesson plans, built a supporting software tool, and made intervention and observation in the actual classroom settings to implement the GLIDE approach.

5.4.1 Participants

27 12th grade students (24 male and 3 female; aged from 17 to 18) at a public high school with a concentration on information technology in Queens, NY participated in this study. They were all students in a Web Client Programming course, which used a series of lab exercises to learn fundamental skills in HTML, CSS, and JavaScript and group projects to build static websites. None of them had experience in using Git, GitHub, or GLIDE before this intervention. The instructor of the course announced the opportunity for voluntary participation and all the course-takers agreed to participate with their parents' consent.

5.4.2 Procedure

We designed an observational field study to identify the learning outcomes and contribution to fair collaboration in web programming in a real-world context. As a DBR approach taking place in actual class sessions at a high school, the main focus of observation is placed on how the proposed GLIDE approach turns out in the real-world settings.

The 27 participants were randomly grouped into 7 teams (6 teams with 4 and 1 team with 3). To better engage the students in their creative work process, the instructor themed the lab activities in the Web Client Programming course as a group competition named “Shark Tank”, where the title of it was adopted from a popular TV show. Each team was asked to come up with an idea for an imaginary technology product and choose two of the other teams as web design agencies competing with each other in building a marketing website for the product. Each team was tasked with two website-building projects (project 1 and 2 in a sequence) as a result of this matching process. The students were told that one of each pair of competing teams that build the marketing websites for the same team (7 in total) will be picked as the winners of the Shark Tank project and they’ll get a bonus credit for their final course grades. The participants were given 2 class sessions for planning and design for each project where each session was 45 minutes in length. They completed each web programming project over 10 sessions. Between the two projects (1 and 2), we used six sessions to introduce the scaffolded feature branch workflow and GLIDE. The students then used GLIDE and the feature branch workflow to collaboratively implement project 2. All of the students had built a static website earlier in the course (project 0) to practice basic HTML, CSS, JavaScript skills. Figure

5.4 illustrates the intervention schedule.

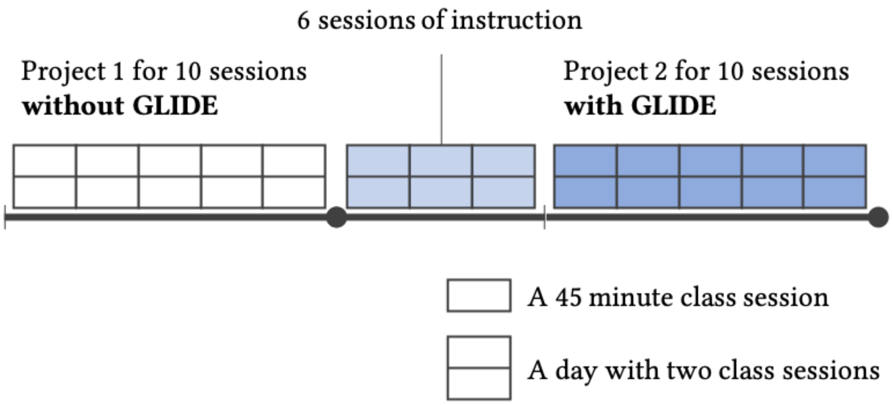

6 sessions of instruction
Project 1 for 10 sessions
without GLIDE
Project 2 for 10 sessions
with GLIDE
A 45 minute class session
A day with two class sessions

Figure 5.4: Procedure for GLIDE Intervention and Observation

5.4.3 Data Collection

We collected data from three types of data sources. First, to identify the learning outcomes throughout the intervention, we audio-recorded the students discussion in the lecture on the scaffolded feature branch workflow from each team between project 1 and 2. This dataset was intended to give us insights on what abstract models they built when they're exposed to the novel approach to collaborative programming for the first time. In the students discussion data, we had two focal points: they were told to explain the concepts on Git workflow they learned in their own language as much as they can; they are encouraged to ask questions within the peer group to clarify their understanding or learn from peers on any topics under collaborative web programming with Git workflow.

Second, we collected the records for coordination and integration process from each team for project 1 and 2. They had no guidelines to follow in integration for project 1, but were told to use the merge feature to integrate individual contribution following Git workflow in project 2. Considering that the GLIDE approach is intended to guide the students to better structured collaboration, this dataset addresses how the participants change their strategies and behaviors in performing the projects, in terms of fairness in collaboration process. The main focus of this dataset is on how balanced the individual contribution is when the group work is divided into individual chunks and incorporated into the shared output as a whole.

Third, we conducted focus group interviews where the participants talk about their overall project experience with or without GLIDE. The interview conversation was recorded and transcribed for analysis. The purpose of the focus group interviews was to get informed about the students' learning experience. Interview questions included the students' perceived value of learning the collaborative workflow and performing a group project following it, and the challenges and limitations of the proposed approach from students' perspectives. Participation in focus groups was optional, but all the participants volunteered to join the interviews with their group members who performed the projects together. We had three interview sessions with two or three groups each. Each session was approximately 25 minutes in length.

As an observational study, we also kept field notes to capture our observation besides the three types of qualitative datasets. One of the researchers attended all the class sessions and took memo about notable phenomena during the lab activities in the classroom. We don't explicitly analyze the field notes as a separate data source, but use the records of what we saw in the classroom over the course of the students projects to provide the analysis results with more context-relevant explanation.

5.4.4 Data Analysis

The coordination records were collected as free form drawings and handwriting on paper, in the participants' project plans for both project 1 and 2. The integration records were collected in a form of online messenger logs for project 1, where the participants copied and pasted their code, sent the image files they designed, compose text content for their websites as text messages. On the other hand, for project 2 where the participants were using GLIDE, the integration records were posted as Git commit history in the project repositories on GitHub. The commit history had all the information on the type of contribution (design, code, text content, project management), the amount of contribution, and how the contribution was incorporated into the group's shared resources. Note that there is asymmetry between the datasets for the integration records for project 1 and 2 in terms of the accuracy and clarity; the integration records for project 1 included at least the cues to infer all the attributes that project 2 yielded, but the amount of contribution is not easily traceable and the way of integration is not visible directly from the dataset. We relied on the field notes to fill this gap with more context-related explanation when needed.

The audio recordings from 21 in-class discussions (3 days 7 groups) and three focus group interview sessions were transcribed and analyzed. We extracted and coded the transcript to get insights into the students' learning outcomes, students' understanding or misunderstanding of the feature branch workflow, their collaboration process, and overall impressions and motivations of collaborative programming.

5.5 Results

5.5.1 Learning Gains through GLIDE Approach

The participants successfully completed the class projects without (project 1) and with GLIDE (project 2). The websites they built as the project outcomes may be the most straightforward evidence that teaching high school seniors how to properly collaborate following a commonly known Git workflow is feasible through lab activities with reasonable levels of preparation and a supporting software tool. To better support the argument, we identified qualitative or observational evidence that the participants picked up the knowledge and skills, but not merely rote-followed the guidance of GLIDE without substantial learning gains. This subsection first illustrates the findings that support the argument on the participants' learning through the GLIDE approach.

In the small group discussion among the participants, they were told to explain the feature branch workflow they learned in their own languages. The transcripts showed that several advanced students were able to build the accurate abstract model about collaboration process using Git workflow, based on their solid understanding of the concepts taught in the lecture. The two examples of the participants' interpretation on the feature branch workflow are presented below.

> (Example 1)
>
> David: "Ted, you start off. Explain everything to us."
>
> Gio: "Save us."

Ted: “So, essentially, what we have now is, we have learned in the past three days, we are learning how to use Git using GLIDE. From that, we have a remote repository that has a master branch that comes default. Since our remote repository is in the cloud, we clone that so we can get a local copy of it. And again, that cloned bit is exactly same as remote except that now it’s in local. Within that local repository, you have your master branch. We can create these things that are called feature branches. Feature branches are just ways for us to add more content to our page. For example, we could add a contact feature branch or a home feature branch, which means, respectively, contact page would be the contact feature branch and home page would be the home feature branch. Then, we have these things called commits, which is just a state of your repository at a given time. Or, it is just a checkpoint in your branch to represent a save.”

David: “To check your progress.”

Ted: “Yes, thanks, David, for adding on to this. Once you make a commit, it is just a save point in your files, which is very important. One important thing to know about content for each of these commits is that all forms of similar content should remain on the same branch. For example, for your contact branch, you should have stuff that represents anything that put for your contact page. Another very important thing to know is that a push is just to upload your given branch to the remote repository to make it all a thing.”

Gio: “Hey, what’s a push again?”

Ted: “A push is to upload. Also, if you wish to, at the end, after you

> have all your commits done to each individual feature branch, you must merge all of them into your master branch.”
>
> (Example 2)
>
> Mark: “Alright, so we have GitHub. We cloned it, to download file, right? You create new branches. And then, after creating branches, you need to work on the branches. Commit and update the branches. Work, so you need to commit again. Once you’re done committing and pushing. And then, you merge the home page branch to the master branch. Add other branches to that master branch. Merge them into the master branch. So that it has all the code in your repository.”
>
> Paul: “So, every time you commit, does it update your local file, or do you have to clone every single time you commit?”
>
> Mark: “That updates your local file. You work on your local file. And then, you push. That updates the remote repository as well.”
>
> Paul: “Okay.”
>
> Eli: “Do you have to put your product page branch in master branch?”
>
> Mark: “Yes.”

These examples show how the in-class discussion went; the most advanced students in each group, usually the group leader, led the discussion by explaining their own understandings and their notes made during the lecture. The other members of the group often followed up with their questions. In those two examples, the group leaders were using the technical terms in explaining the essential steps of the feature branch workflow and the other group members seemed to follow the explanation. This shows the feasibility and learnability of the Git workflow as a course material through collaborative learning.

In the in-depth analysis of the small group discussion and the focus group interview datasets, we were able to also find the misconceptions, misuse of the terms, and a lack of clarity in understanding that the participants had in learning Git workflow. Table 5.2 shows the frequency of the misconception identified through a content analysis of the students discussion logs on the feature branch workflow. Note that this part of results reflects the participants' conceptual understanding on the subject because the lecture and discussion occurred before their project 2 using GLIDE during which they actually performed actions for the Git-related operations on their computers for the first time.

As the participants were told to discuss what they learned at the end of the sequential lecture sessions for the three days, it seems natural that the topics covered earlier (e.g., repository and clone) were referred more frequently than those covered later (e.g., commit, push, and merge) in total. It's also understandable that the largest portion of the participants discussion was around the branches, master and feature branches combined, since the branches are the main unit of operation in the feature branch workflow. We couldn't find the empirical evidence of participants' misconceptions or questions on local repository, while merge seemed to be the most challenging for the participants to understand.

Table 5.2: Content analysis results on the frequency misconception identified in the feature branch workflow

Terms for Git Workflow		Reference Count	Misconception Count	Example of Misconception or Question	Types of Misconception or Challenges Identified
Repository	Remote	15	1	"What's the difference between Git and GitHub?"	· Distributed model with the central repository
	Local	18	0	N/A	N/A
Clone		24	2	"We can download the remote repository as a zip file. Why do we clone?"	· Metadata for tracking history
Branch	Master	13	2	"Does the team leader make the master branch?"	· The origin of master branch
	Feature	18	3	"So, basically, it allows the group members share each other's files because everyone downloads the files?"	· Asynchronicity · Branch-level work unit
Commit		16	3	"So, every time you commit, does it update your local file, or do you have to clone every single time you commit?"	· Asynchronicity · Distributed model with the central repository
Push		5	1	"Hey, what does a push mean, again?"	· Skipped often
Merge		15	7	"Do you have to put your product page branch in master branch?"	· Automatic process of fast-forward merge · Asynchronicity · Branch-level work unit · Distributed model with the central repository

5.5.1.1 Branch-Level Work Unit

One of the most frequent errors in understanding the feature branch workflow we identified in the dataset was rooted in the understanding on a branch as a unit of work, rather than files and folders. Because a branch is an abstract concept, the participants needed to adjust to the new way of thinking. This example of students discussion below shows the learning gain required to understand a branch as a work unit in the feature branch workflow.

> Adam: "We have the master branch and we all make feature branches."
>
> Beck: "Does the team leader make the master branch?"
>
> Adam: "No, we already have it. We make feature branches, and then, we make commits."
>
> Carlos: "What exactly is... A branch is like... Is it like a folder? If the master branch holds files, then it's like a folder, essentially, I guess."
>
> Dana: "Is this easy for you guys?"
>
> Adam: "Yeah, probably."
>
> Carlos: "No."
>
> Dana: "No, right?"
>
> Beck: "This is so much more difficult than what we did before."
>
> Adam: "And then, we make commits and merge them together. And it makes the whole page."
>
> Beck: "Okay, so everyone gets a feature branch. We assigned a role. We keep committing, just like a checkpoint. And then, we're done. And then, we merge."

The participants had the basic computer skills and literacy to handle files and folders. Branches represent another flexible abstraction of a collection of files and folders in the repository so that a user can dynamically switch between, where each branch may have different content according to the author and purpose. Since the participants haven't been exposed to this concept or used a branch, the notion that "a branch has files and folders you are working on" might have brought about the misconception that a branch is another type of a container, such as a folder. Technically, Git manages multiple branches by dynamically keeping track of the changes in the file system, so Carlos's interpretation of a branch as a folder-like static container is inaccurate. However, from the user's perspective, it also makes sense that each branch, as a top-level container for files and folders, contains different contents where it has an option to automatically merge into another. This result shows how the participant interprets the novel concepts in Git workflow from the technical vocabulary and skills they already have. Carlos's reference in the focus group interview was encouraging in the sense that the learning in the Git workflow was challenging enough for the participants to stretch their way of thinking at manageable levels when they had a chance to actually apply those concepts in the lab activities.

> Carlos: "It was really hard in the lecture, but when we did the project, it turned out to be new terms developers use, but it's pretty much what we do. We change file names, we keep separate folders not to mess up on the colleagues' files, and it's like, we look for the changes in files to update those files. You have the tools to do all this stuff on GLIDE and GitHub, so it's really convenient."

He closely related the essential steps of the feature branch workflow to his prior experience in working on group projects. This quote from Carlos is another piece of evidence supporting that the participants were able to build conceptual models of the proposed work process, rather than they relied on how the software tool guides them by rote. In addition, he acknowledged the technical terminology is used by professional software engineers and developers and actively mapped what is means in terms of what he did for his own project.

5.5.1.2 Asynchronicity

The most frequently repeated misconception seemed to stem from asynchronicity, which means two or more pieces of individual contributions are made at different times; not in a synchronized fashion. This misconception was found in relation to feature branch, commit, and merge. This excerpt from the students discussion below show how participants might have misinterpreted those Git operations due to the unfamiliarity with asynchronicity.

> Matt: "So, basically, from merge, you can put all the progress?"
>
> Rick: "Each of us works on a different section of the repository."
>
> Matt: "Like you can take like, you being master branch, him being the whole product page, and I being the about page. All the progress we made was uploaded to you."
>
> Rick: "Yup."
>
> Matt: "If he committed, he saved his progress."
>
> Rick: "Right. He can also commit on some other branches. Then, that branch goes to master."
>
> Ralph: "Does it allow you to share the progress?"
>
> Matt: "I'm not sure. Does it?"
>
> Rick: "It does, because every commit made on the other branches goes into the master. So, master has every single commit on it."

> Matt: "So, basically, it allows the group members share each other's files in right away, because everyone downloads it?"
>
> Rick: "No, they're working on separate branches at different times."
>
> Matt: "Okay. So, it allows each of the members to do one single task."

In the discussion on feature branches and merge among Matt, Rick, and Ralph, Matt had clear understanding on feature branches as a separate unit of work coordination (underlined). However, he didn't fully understand how merge happens (underlined) without understanding asynchronicity of the workflow. The group leader, Rick, was able to correct him that commits on feature branches happen at different times and merge doesn't happen in real-time. This misconception was also observable in focus group interview. Kate talked about her attempt to look up others' work in performing the group project using GLIDE.

> Kate: "I wanted to see what he's doing and tried to match the color for the page I was working on. So, I went to his branch, but nothing was there. I didn't see anything he added to the template at that moment. I found it on master branch later, after he finished his work."

Checking collaborators' progress and making design attributes consistent is an essential element in building websites as a group, but her approach to doing that reveals she was presuming that the progress made by others is shared in real-time, which is a misconception on asynchronicity of the feature branch workflow. In this case, she was able to learn the updates become available once the others' feature branch is pushed and merged into master branch.

5.5.1.3 Distributed Model with the Central Repository

One of the key ideas of the feature branch workflow is that each collaborator can work one's own local copies to make the potential merge into the shared resources (master branch) for the incremental updates. This model is tied to a specific architecture involving a remote repository (origin or upstream) and local repository with feature branches where these repositories interact with each other through Git operations. This model created a novel settings for the participants; we observed that they have been sharing their files in peer-to-peer using online messengers, as file attachments or as copied-and-pasted text chat messages. In the focus group interview, the participants talked about the differences between the peer-to-peer sharing and the distributed model with the central repository that the Git workflow employs. Even though this project was the first time for them to work with the distributed model with the central repository, they commonly said this was helpful in incorporating their individual contributions.

> Jesse: "It was very messy when we send the code on Slack. I just don't know where the code goes when I received the file from the team members. And I request the files over and over again when I have no idea what's the right one to copy or how to put them all together in the right way."

> Ted: "The most memorable thing for the project was, it's like merging all the code. It was actually the first time we knew how to merge all we have. In DreamWeaver, we did not really know how to do it. I really liked the feature where... when you put everything together

> into the master, in one place, and you make another branch, then you have everything already. Putting everything in one place makes it very simple."

These examples show that the different model of settings that the Git workflow requires a certain level of adjustment built on top of their prior experience, which is favorable for an educational purpose, since it helps them share resources in more organized ways and reduce the potential errors, according to the participants.

5.5.2 Utilitarian Benefits of Using GLIDE

In addition to the learning gains from the lab activities using GLIDE, the participants also reported on the utilitarian benefits of using it through the project report submissions and the focus group interview.

5.5.2.1 Deployment and Handling the Remote Resources

As a web-based application that communicates with project repositories on GitHub, GLIDE is designed to access user data including website resources on the remote server. In addition, GitHub pages simplifies deployment of website into a single configuration item. This application architecture and configuration support very easy deployment and handling of remote resources. The participants acknowledged the usability of this feature as one of the direct benefits they can get from GLIDE.

> Adam: “You don’t have to upload files to your private server. You don’t have to connect to the FTP. Everytime you commit and push, you upload them.”

> Trea: “I would say uploading would be tedious, something like that, because the file names and stuff like that without GLIDE. Because GLIDE is like simple. DreamWeaver is usually easy, but if you’re using it for the first time, it’s not.”

This utility resonates with the teachers’ needs we identified in chapter 4 that deployment requires technology supports for novice students as it involves the basic background knowledge on how to deal with web servers, which can hardly be assumed for them to have. Even though the dataset collected only from the students doesn’t explicitly reveal how teachers would like it, the teachers’ needs we observed in chapter 4 suggest that this feature would also help them reduce the managerial cost in teaching.

5.5.2.2 Better Organization of Project Resources

A branch as the unit of work appeared to have assisted in organizing the participants’ project resources. It makes clear distinction of one’s work from the others, encapsulates relevant files and folders, and labels the work divided for individual contributions.

> Bruce: “I’d say I felt the workload is less because it’s all separated into branches. If not, it’s too much to work on.”

> Kaleb: “There is also that part where that’s like time management for work efficiency. We know when we need to get things done. We learned to manage our time much better.”

Project 2 and project 3 tasked the participants with the same lab activities with the same levels of requirement, so there are no clear reasons to say the workload actually reduced. One possible interpretation is that Bruce’s comment on the perceived workload implies the participants might have benefited from better organization of project resources that the Git workflow of GLIDE approach provided. Moreover, Kaleb’s comment implies that this practical benefit might have also improved their collaboration process by clarify the tasks already done and to do next for the collaborators. Work efficiency and time management that Rick mentioned were not the direct goal this study tried to achieve through GLIDE approach, however, the way GLIDE and the feature branch workflow transform student tasks suggested this positive impact on their collaborative lab activities.

5.5.2.3 Tools for Peer Review

The participants reported that the combination of GLIDE and GitHub provided usable and useful tools for peer review, which they think improves the quality of their work.

> Rick: “We’ve done separate pages on our own. Sometimes, we need someone to look over our code and ‘oh, you have a mistake here. Fix this and that’, a peer edit, basically. You’re advised to make your pages better.”

> Interviewer: “What was the most memorable thing about project 2?”
>
> Derrik: “For everyone in my group, including me, it was nice that everyone actually contributed. We communicated [on GitHub] and used GLIDE.”

The specific tools that the participants used for peer review include pull request to notify when they’re ready for a review (on GLIDE and GitHub) and the merge conflict editor (on GitHub) to make changes in the shared code. In project 1 without GLIDE, review was only possible by visiting another collaborator’s computer without any systematic tools support. This also seems to be a practical support for peer review or peer instructions for teachers, judging from the teachers’ report that they actively employ peer instruction to teach lab activities in chapter 4. Derrik’s comment shows that the combination of GLIDE and GitHub worked well as tools for peer-driven code review. They were able to communicate using the editor on GitHub in resolving merge conflict for integration, while writing individual code on GLIDE. We argue that our approach presented a rare example of natural integration of peer-driven code review for in-class collaborative lab activities.

These examples show the practical assists in performing lab activities that the participants were able to directly get from GLIDE approach and the Git workflow. This supports the argument that GLIDE approach is feasible to implement in actual classrooms without demanding too much to the students in terms of making adjustments in the way they perform the tasks. As a scaffolding tool for novice students, GLIDE seems to have provided the sufficient affordances and ease-of-use as well as the effectiveness in the task performance.

5.5.3 Fair Contribution in Collaborative Web Programming

We also investigated how the GLIDE approach contributes to fostering collaborative learning environment in a CS classroom. From the coordination and integration records of the participants, we observed that this approach helped the participants fairly contribute to their collaborative group projects in a CS classroom context.

The first evidence for the GLIDE approach contributing to the fair contribution in collaborative web programming was found during the observation of how they coordinate the work. The project plans and designs that each group submitted included their coordination plans in the early stages of each project. In project 1 with no guidance for collaboration process, an imbalance of types of tasks was commonly found across the groups; all the seven groups had at least one member who was not planning to contribute to any coding task at all, but graphic designs for image resources using Photoshop application. The participants might have different specialty skills in different areas or interest, but work coordination for a collaborative in-class project must not allow them to avoid a specific type of task. Making this imbalance even worse, all of the three female students in this class were those dedicated designers. Those designer students usually created logo designs, edited background images, and generated color schemes for other members using Photoshop. Creating image assets is a necessary task in building a website, however, it's hard to justify that the designer students solely focus on this task for the group project in the Web Client Programming course for fulfilling the learning goals of this course. In sum, project 1 without guidance in the coordination and the way they work as a group left the participants to merely stick to one's preferred types of task, which brought about the imbalance of tasks as a result.

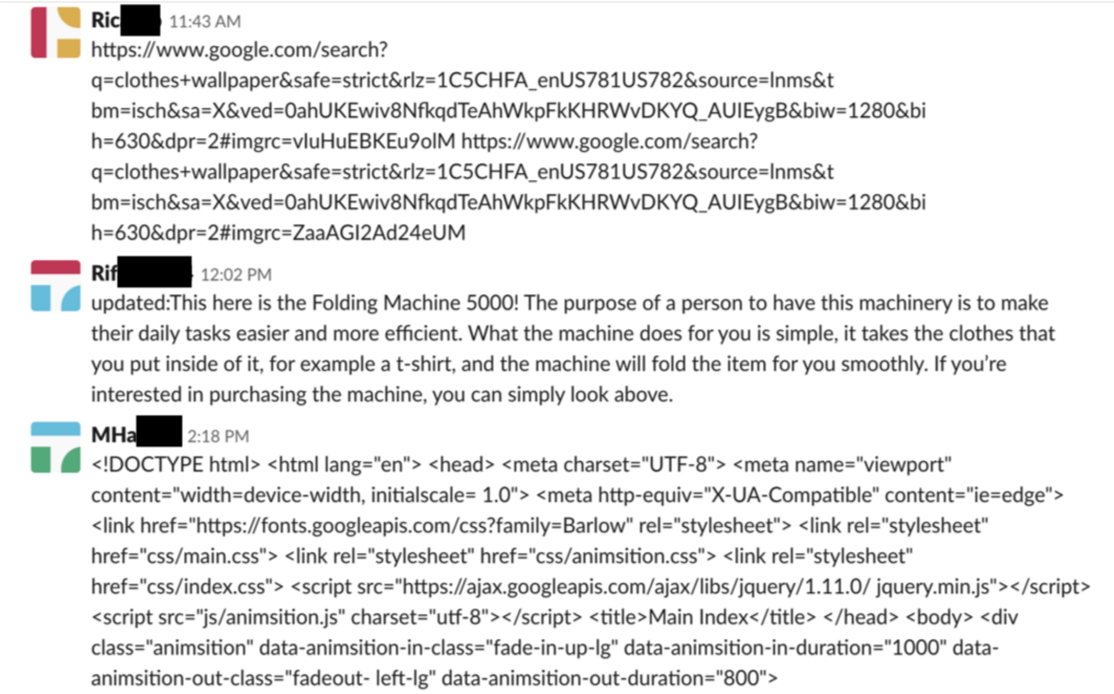
On the other hand, the coordination plans the participants submitted for project 2 had two noticeable changes from those for project 1. First, their coordination plans were described in more concrete detail, in terms of who's going to be responsible for what tasks. In project 1, six out of the seven groups submitted the coordination plans with very abstract description of tasks (e.g., coding and design) or even unclear position titles (e.g., CEO and product manager) without any further explanation on what tasks they were going to perform. In project 2, however, six out of seven groups came up with more concrete coordination plans, where each member's role is specified at page level or feature level. Second, the imbalance of tasks was mitigated in terms of the dedicated designer role. The coordination plans for project 1 showed at least one dedicated designer, who's not supposed to work on HTML, CSS, or JavaScript, per group, but design graphical resources using image editing tools. However, their coordination plans for project 2 didn't have any dedicated graphic designers, which means everyone in a group had to contribute to coding tasks in HTML, CSS, or JavaScript. Table 5.3 shows how the role descriptions of the dedicated graphic designers have changed, based on the coordination plans the participants submitted in the early stages of each project.

Along with the "dedicated designers", another point of the observation that characterizes the imbalance of contribution in project 1 was manual code integration by the group leaders. In project 1, each group member copied the code he wrote and pasted it on an online messenger (Slack). The group leader receives the message, copies the code segment, and paste it into his own copy of the website codebase (Figure 5.5). In doing so, figuring out where the code

Table 5.3: The changes of the removal of dedicated graphic designers identified in the coordination plans

Team	Changes in the Roles of Dedicated Graphic Designers	
1	Project 1	**Oscar: Creating logo and banner**
	Project 2	Oscar: Creating images and coding CSS
2	Project 1	**Paul: Designs** **Eli: Designs**
	Project 2	Paul: Buying the Product page / Photoshop images Eli: About Us page / Lead designer - CSS
3	Project 1	**Rick: Designer**
	Project 2	Rick: JavaScript
4	Project 1	**David: Main designer**
	Project 2	David: Designing the products / coding Buy page
5	Project 1	**Job: Graphic designer** **Kevin: Graphic designer**
	Project 2	Job: Web designer for Home page Kevin: Web designer for External Services Link page
6	Project 1	**Dana: Logo design**
	Project 2	Dana: Logo design and coding Product page
7	Project 1	No coordination plans specified **(Observed Perri, a female student, being the dedicated designer in this group)**
	Project 2	Perri: Forums page

segment goes, which specific part has been updated, and how to make the code work with other parts are solely left onto the group leader's responsibility. These tasks are the additional work to individual commitment that group projects essentially involve, which means the group leaders are burdened to be in charge of the integration tasks. On the other hand, the merge operation of the feature branch workflow makes the integration process more accessible to every member in a group and automate the tedious repetition of identifying updated piece of code, making comparison between old and new versions of it, and replacing old code segment with the new one. Thus, the merge commits made by non-group leaders are the evidence that supports the GLIDE approach supports better balanced code integration process in a group project. Figure **??** shows the diverse patterns of merge of code made by different participants.



Ric[redacted] 11:43 AM
https://www.google.com/search?
q=clothes+wallpaper&safe=strict&rlz=1C5CHFA_enUS781US782&source=lnms&t
bm=isch&sa=X&ved=0ahUKEwiv8NfkqdTeAhWkpFkKHRWvDKYQ_AUIEygB&biw=1280&bi
h=630&dpr=2#imgrc=vluHuEBKEu9olM https://www.google.com/search?
q=clothes+wallpaper&safe=strict&rlz=1C5CHFA_enUS781US782&source=lnms&t
bm=isch&sa=X&ved=0ahUKEwiv8NfkqdTeAhWkpFkKHRWvDKYQ_AUIEygB&biw=1280&bi
h=630&dpr=2#imgrc=ZaaAGI2Ad24eUM

Rif[redacted] 12:02 PM
updated:This here is the Folding Machine 5000! The purpose of a person to have this machinery is to make their daily tasks easier and more efficient. What the machine does for you is simple, it takes the clothes that you put inside of it, for example a t-shirt, and the machine will fold the item for you smoothly. If you're interested in purchasing the machine, you can simply look above.

MHa[redacted] 2:18 PM
<!DOCTYPE html> <html lang="en"> <head> <meta charset="UTF-8"> <meta name="viewport" content="width=device-width, initialscale= 1.0"> <meta http-equiv="X-UA-Compatible" content="ie=edge"> <link href="https://fonts.googleapis.com/css?family=Barlow" rel="stylesheet"> <link rel="stylesheet" href="css/main.css"> <link rel="stylesheet" href="css/animsition.css"> <link rel="stylesheet" href="css/index.css"> <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.0/ jquery.min.js"></script> <script src="js/animsition.js" charset="utf-8"></script> <title>Main Index</title> </head> <body> <div class="animsition" data-animsition-in-class="fade-in-up-lg" data-animsition-in-duration="1000" data-animsition-out-class="fadeout- left-lg" data-animsition-out-duration="800">

Figure 5.5: Sharing code and website content on an online messenger for manual code integration by the group leader

As a follow-up analysis, we also observed how well-balanced the participants actual contribution was in project 2. GitHub has a built-in feature called "insight" where the statistics of all the contributions from the collaborators are aggregated and visualized in plots (Figure 5.7). The contribution statistics include how many code lines were added, how many code lines were deleted, and how many commits were made by each contributor to the final outcomes

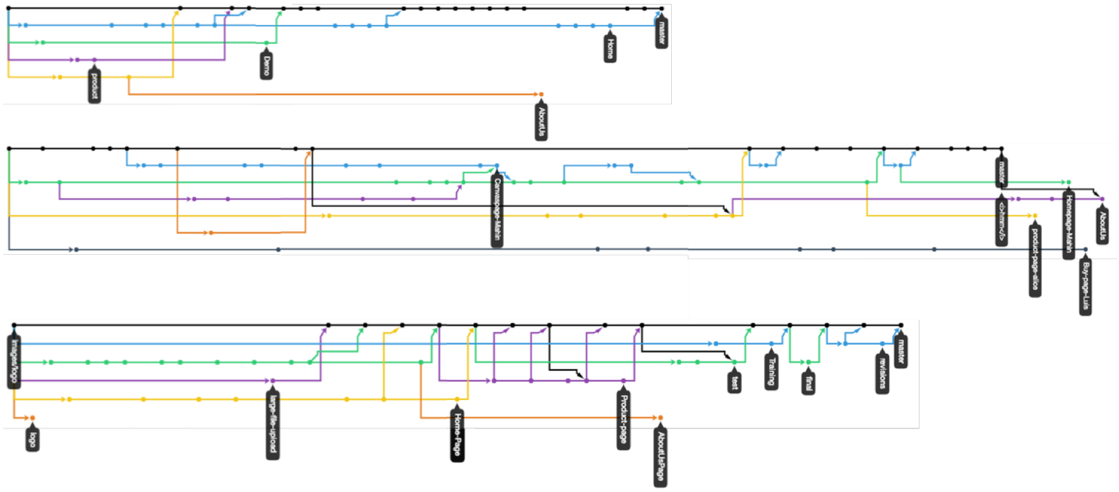
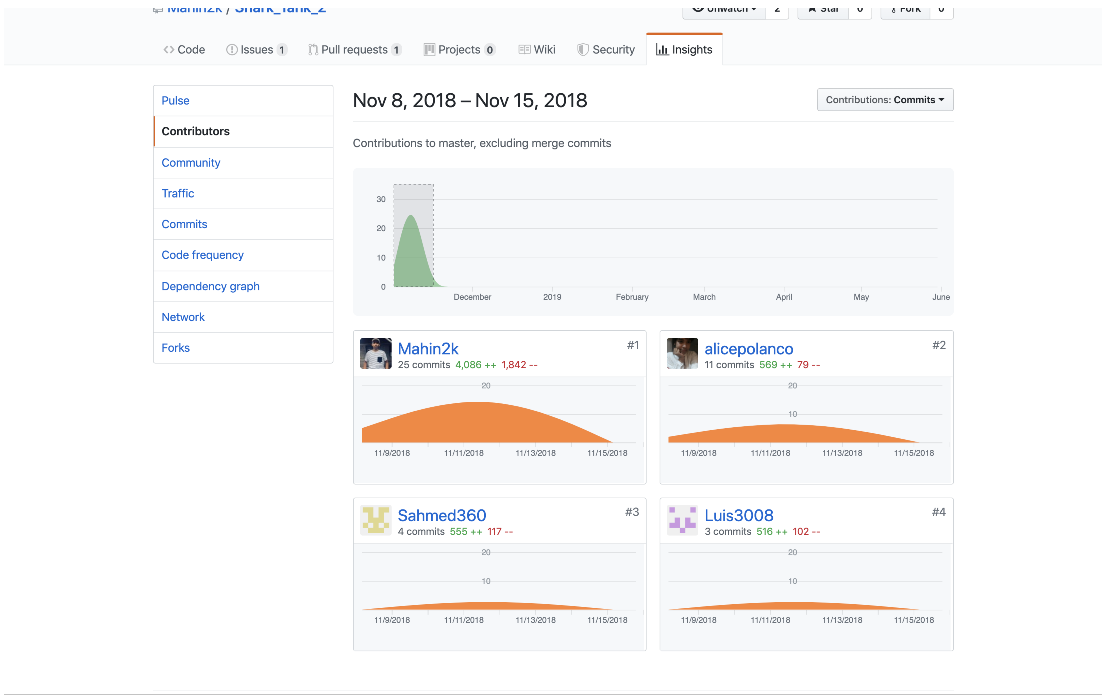


Figure 5.6: Examples of commit trees showing that all the members can work on code integration following the Git workflow

Table 5.4: Contribution stats per group from Git log

Team	Number of Commits per Member	Code Lines Addition per Member	Code Lines Deletion per Member
1	5 / 3 / 3 / 3	531 / 509 / 495 / 193	71 / 42 / 99 / 29
2	25 / 11 / 4 / 3	4086 / 569 / 555 / 516	1842 / 79 / 117 / 102
3	5 / 2 / 2 / 1	672 / 315 / 281 / 159	217 / 193 / 132 / 53
4	29 / 6 / 4	3591 / 369 / 11	1658 / 10 / 5
5	17 / 4 / 2 / 1	641 / 243 / 18 / 46	557 / 4 / 9 / 12
6	14 / 10 / 2 / 1	304 / 333 / 152 / 56	104 / 212 / 14 / 20
7	15 / 10 / 1 / 1	424 / 1188 / 100 / 248	544 / 623 / 8 / 5

on master branch in a repository. Given that the GLIDE approach in project 2 alleviated the two phenomena regarding the imbalance into the group project, all the seven groups showed acceptable level of balanced contribution across the group members in terms of the number of commits, number of addition of code lines, and number of deletion of code lines (Table 5.4).



Nov 8, 2018 – Nov 15, 2018

Contributions to master, excluding merge commits

Mahin2k
25 commits 4,086 ++ 1,842 --

alicepolanco
11 commits 569 ++ 79 --

Sahmed360
4 commits 555 ++ 117 --

Luis3008
3 commits 516 ++ 102 --

Figure 5.7: Repository "insight" visualizing the participants activity during the project period on GitHub

5.6 Discussion

The results presented above support two arguments that this chapter makes: 1) the GLIDE approach might provide an effective learning-by-doing opportunity for novice students with well-structured collaborative process through group projects in a web programming course and 2) the design of the GLIDE approach might encourage better balanced contributions of the students within a collaborating group. In this section, we add further explanation on the findings based on the observation.

The findings regarding the learning gains through GLIDE rendered the potential for more collaborative CS education. While using version control systems in programming activities has been considered as an advanced users' skill, our observation that supports the learnability of the proper collaboration process sheds light on introducing the tools and curriculum into novice levels. Also, we argue that the difficulties and misconceptions the participants faced in our observation might suggest more of an opportunity to more actively pursue this approach, largely because the students were able to resolve and fix the errors through the in-class discussion with the help from another student. In addition, the sources of misconceptions, such as branch-level work unit, asynchronicity, and distributed model of server and clients, might address significant and relevant topics to cover. For example, understanding branch as a work unit involves abstraction, which is one of the most crucial elements of computational thinking; asynchronicity in the distributed model renders a good chance to cover learning the fundamental web architecture, embedded in the lesson for the collaboration process.

In terms of fairness in collaboration, the design of the context and GLIDE approach seemed to have the participants behave differently in terms of coordination and integration. Then, what did lead the participants to different behavior with GLIDE? One possible explanation on how GLIDE structured their behavior in better balanced ways is that the branch structure and the user interfaces of GLIDE and GitHub made their work division clearer, which in turn led to fair coordination. After project 1 finished, the participants learned how the work coordination is embedded in the feature branch workflow during the three days of lecture; each branch may represent a unit of tasks, the software tools (GLIDE and GitHub) show who created the branch, each commit on a branch shows who made the commit, the commit history makes their work progress traceable, and a commit has the summary of the task completed through commit messages. Furthermore, the participants had a chance to actually try those features during the practice time at the end of the lecture on each of the three days. When they were told to plan and design their project as they did for project 1 on the first day of the project 2, they might have chosen the more explicit distinction among the tasks and relatively more concrete work definition, as they learned how branches work. The user interfaces on the software tools have labels of who created the branch and who made the commits. Also, the participants used branch names as the identifier for a feature or a page to build on the branch itself. This improved clarity of coordination might have led to the better balanced coordination.

This explanation resonates with existing research on task awareness and group awareness in CSCL, which are known as effective mediators that help successful coordination. Task awareness refers to information on the materials and strategies needed to successfully perform the task [45], while group awareness means information on the states and expected behaviors of team members individually or as a whole [51]. Those features of GLIDE and GitHub mentioned above, clear separation of work division, explicit labeling for the work divided, and the potential learning from the lecture and practice before project 2 seemed to have provided better cues for task awareness and group awareness.

We conducted follow-up analysis to identify qualitative evidence supporting the explanation that the participants perceived increased task awareness and group awareness with GLIDE approach, which might have in turn changed their behavior during the lab activities in project 2. From the focus group interview data, we were able to observe a participant's reference on task awareness as follows.

> Ralph: "In project 1, we were almost like more of whatever we can do, just get it done, but for project 2, we had to follow the guideline using GLIDE."
>
> Interviewer: "What do you mean by the guideline?"
>
> Ralph: "We had to work on GLIDE. Actually, for project 2, we were more organized. That's what I want to say."

Compared to "Whatever we can do" implying ill-defined procedure of the tasks, the "guideline" of GLIDE approach seems to have given him more information on the task. Even though we weren't able to directly measure how much task awareness the participants had for each project, this example reference suggests the possibility of the workflow having contributed to the increased awareness of the materials and strategies for the tasks.

The reference by Alice in section 6.4.2, saying that "If you want to see someone else's work in your team, you could just look into their branches", is a good example reporting on her perceived group awareness. She was aware of other collaborators' roles and able to see what they were working on. In this specific case, branch structure and the labels worked as the cue for the awareness that informs her about the tasks that the collaborating group is working on.

The proposed lesson plans and GLIDE made the Git workflow more approachable by providing the adequate level of explanation, rather than simply abstracting away the technical fundamentals of how to work with others on web programming. Reflecting on Vigotskian learning theory [127], the proposed approach tried to implement the better contextualized learning in three ways: first, in the process of structured collaboration, GLIDE scaffolds the difficult Git operations (e.g., command line tools) to bring those difficult knowledge down into the students' ZPD (zone of proximal development), where they can learn through interaction with others; second, the group project context we designed for the DBR actively involved the interaction with MKO (more knowledgeable others), which was the instructor and the other group members, including in-class discussion and communication during the lab activities; third, our approach actively involves the terminology, norms, and tools widely used in the field of software engineering, which helps indirect interaction with the broader society, according to the theory.

On a practical side, as pointed out in Related Work section, collaborative programming using version control systems has been left out of the curriculum for young students. This gave novice students almost no options in tools and courses to systematically learn how to properly work with others in learning CS. The lesson plans we designed and GLIDE to support the lesson plans will help teachers and students with the widely used collaboration process.

5.7 Summary

The existing research has been using Git as a courseware or tools for lab activities, but little has been reported on teaching novice students how to collaborate on the foundation of Git workflow. From the propositions made in the previous chapter, which calls for a better structured and effective way of working with others, we implemented a classroom design with lesson plans on the feature branch workflow and a software tool, GLIDE, that students can build websites as a group following the workflow. We took DBR approach to propose and deploy GLIDE approach, where students took a series of lectures on the feature branch workflow and performed group projects applying the skills and knowledge they learned. The observation and qualitative data collected from this intervention showed that, unlike the general perception of teachers on Git, the participants were able to understand and apply how to effectively work with others on the feature branch workflow using GLIDE, and consequently, build their group websites with no serious issues. The students discussion log revealed that they had misconceptions in branch-level work unit, asynchronicity in making progress, and distributed model with central repository. However, the discussion data also showed that the participants' peer discussion was enough to correct those errors. Furthermore, our approach emerged as a potential measure to improve fair contribution among group members, by increasing their awareness of the traceability of their work and explicitly clarifying their work definition through branch structure.

CHAPTER 6

STUDY 4: QUANTITATIVE EVALUATION OF STUDENTS' LEARNING EXPERIENCE WITH GLIDE

6.1 Motivation

In the previous chapter, we proposed the lesson plans and GLIDE to teach novice students how to effectively collaborate with others in compliance with the widely used Git workflow in class project settings. To get better informed about the consequence and influence of the proposed approach, this chapter report on the quantitative evaluation of students' learning experience with GLIDE. We take a field experiment approach to the evaluation as part of the intervention we reported in the previous chapter, rather than conducting an isolated and decontextualized lab experiment for a usability study, to better understand the students' experience in the actual learning context. In this chapter, we report on the hypotheses testing on the effects of proposed approach to collaborative workflow using GLIDE on students' learning experience from a field experiment using survey questionnaire. In sum, we provide insights into how to better teach collaborative programming through our proposed approach and an empirical study as an evaluation of the proposed approach.

6.2 Hypotheses

We generally hypothesized that novice students report more positive learning experience when they performed web programming projects following the scaffolded feature branch workflow using GLIDE compared to when they did without the guidance. Since learning experience is a vague concept mingled with diverse elements, which makes it hard to evaluate or operationalize, we focus on three attitudinal and experiential outcomes of learning, drawn from existing literature. First, we hypothesize that performing project with the guidance of the scaffolded feature branch workflow leads to higher levels of student engagement [106]. Student engagement has been discussed as a crucial factor for successful school completion and well-motivated learning [107]. Shernoff et al. [106] argued that reforming student-driven activities in classroom to support an appropriate level of challenge for students' skills increases student engagement. In this research, the proposed approach to collaborative web programming project using GLIDE scaffolds the hard tasks of Git workflow, so that high school students can perform the project. Also, the proposed lesson plans provide the terminology and conceptual model of the workflow, where students can develop sufficient levels of skills for collaborative programming tasks. Thus, the scaffolded feature branch may contribute to increased levels of student engagement.

The second hypothesis is that the proposed approach to the collaborative web programming project brings about students' higher levels of psychological ownership [121] in projects. From organizational behavior perspectives, psychological ownership in work context has been a significant factor for work performance and organizational citizenship, that is correlated with a sense of responsibility in work environment [121]. In this research, students are supposed to work on their own feature branches according to the way Git workflow organizes individual units of collaborative work. This better structured way of distributing and assigning individual's work unit may contribute to increased levels of psychological ownership in student project.

Lastly, we hypothesized that the proposed approach increases students' perceived fairness [42] in contribution and acknowledgement in performing group projects. One of the most frequently reported difficulties in collaborative learning in education is accurate and fair credits for individual performance [42]. The feature branch workflow explicitly visualizes individual's contribution by showing the contributor's user name and what and how much code the user contributed. This resolves the ambiguity of expected credit for oneself and for other collaborators, which contributes to increased levels of perceived fairness in contribution and acknowledgement.

To sum up, we hypothesized that the scaffolded feature branch workflow using GLIDE in collaborative web programming projects leads to:

H1 : higher levels of student engagement

H2 : higher levels of psychological ownership in project

H3 : higher levels of perceived fairness in contribution and acknowledgement

To test these three hypotheses, we conducted a field experiment in the actual education settings as follows.

6.3 Methodology

6.3.1 Participants

The same group of students who participated in the qualitative study in the previous chapter joined this survey study. They were 27 12th grade students (24 male and 3 female; aged from 17 to 18) taking Web Client Programming course at a public high school with a concentration on information technology in Queens, NY. They had the basic skills in HTML, CSS, and JavaScript through a group project to build static websites before this intervention. This intervention was their first experience in using Git, GitHub, and GLIDE.

6.3.2 Procedure and Experimental Design

We designed a one-way within-subject field experiment to examine the effects of the intervention in a real-world context. One of the reasons to have chosen the one-way within-subject A-B-A design over a more conventional between-subject design with treatment and control groups was because of practical limitations in finding comparable class for the control condition, where the instructor of the class agrees to participate. Another reason was to separately investigate the treatment effects and time-based effects through repeated measurements; using A-B-A design with repeated measurements, it becomes possible to separate the gains or losses in outcome variables explained by time (or the iteration of projects from project 1 to project 3 in this research) from the treatment effects.

The grouping and matching scheme was the same as the observational study in the previous chapter; they were randomly grouped into 7 teams (6 teams with 4 and 1 team with 3); each team came up with an idea for an imaginary technology product and was tasked with two website-building projects (project 1 and 2 in a sequence). The participants performed each web design project over 10 sessions where each session was 45 minutes in length. The participants took lecture on the feature branch workflow for six sessions between the two projects (1 and 2). Once they completed the project 2, they moved on to the next website-building project (project 3) of the course without our intervention. Figure **??** illustrates the procedure.

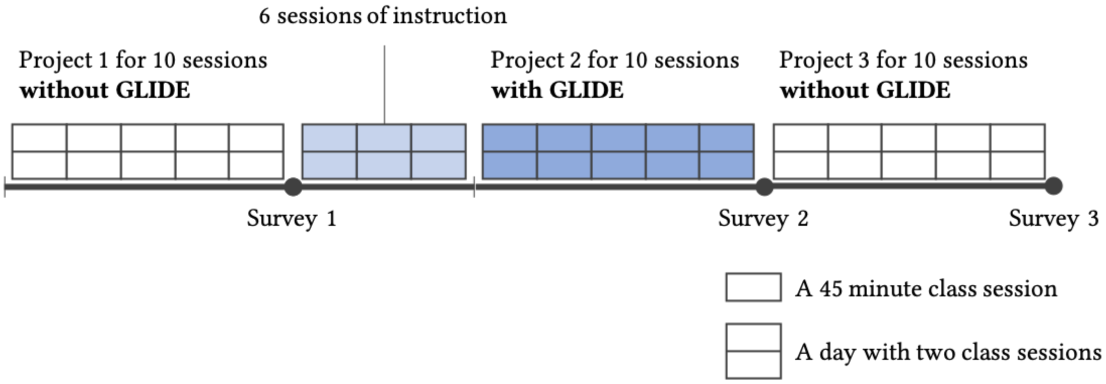

6 sessions of instruction
Project 1 for 10 sessions
without GLIDE
Project 2 for 10 sessions
with GLIDE
Project 3 for 10 sessions
without GLIDE
Survey 1
Survey 2
Survey 3
A 45 minute class session
A day with two class sessions

Figure 6.1: Procedure for intervention and survey

6.3.3 Data Collection

We conducted three iterations of repeated measurements of a survey questionnaire in A-B-A design: survey 1 after project 1 (no guidance in workflow), survey 2 after project 2 (the feature branch workflow on GLIDE), and survey 3 after project 3 (no guidance in workflow). The questionnaire used 15 items (5 items for each variable) to ask the participants' self-reported student engagement, psychological ownership in project, and perceived fairness in contribution and acknowledgement during the group projects, using a 7-point Likert scale. These measurement items were either adopted from existing literature on each outcome variable or newly composed to fit in the group project context and the age levels if necessary. Table 6.1 shows the measurement items used for the survey.

After all the surveys, we conducted focus group interviews with the participants to get informed about the students' learning experience that the predefined survey questionnaire cannot capture. Interview questions included the students' perceived value of learning the collaborative workflow and performing a group project following it, and the challenges and limitations of the proposed approach from students' perspectives. All the participants chose to join the interviews with their group members who performed the projects together. They joined the three interview sessions with two or three groups each and each session was approximately 25 minutes in length.

6.3.4 Data Analysis

The dataset consisted of 81 questionnaire responses (27 participants 3 repeated measurements) in total. We analyzed the data with a linear mixed-effects model approach, which is an extension of linear regression, involving the estimation of fixed effects (the treatment and time) and random effects (participants). This approach leads to less unexplained variance and more statistical power than traditional approaches, such as repeated measures analysis of variance, because

Table 6.1: Survey questionnaire

No.	Outcome Variable	Measurement Item	Note
1	Student Engagement	I was concentrating on the project.	Shernoff et al. [106]
2		I found the project activity interesting.	Shernoff et al. [106]
3		I enjoyed the project activity	Shernoff et al. [106]
4		I was engaged in writing code or designing images for the project.	Newly composed
5		I was engaged in discussion with the team members.	Newly composed
6	Psychological Ownership	This is "MY" project.	Van Dyne and Pierce [121]
7		I feel a very high degree of my ownership for this project.	Van Dyne and Pierce [121]
8		I sense that the outcome website is "my" website.	Van Dyne and Pierce [121]
9		The team members and I feel as though we own the project.	Van Dyne and Pierce [121]
10		I feel I invested a lot in this project.	Newly composed
11	Perceived Fairness	All the team members developed key parts of deliverable.	Fellenz [42]
12		All the team members showed "good citizenship".	Fellenz [42]
13		I feel that the amount of the work across the team members was well-balanced.	Newly composed
14		I can easily identify everyone's contribution in the project outcome.	Newly composed
15		Everyone in my team had fair share of workload.	Newly composed

it doesn’t aggregate observations for a unit of analysis (a participant), which results in a loss of information [112]. Using this statistical model, we examined the significance of the model using p-value and the size of effects using Cohen’s d. We used lme4 package [13] in R [117].

The focus group interview recordings were transcribed and analyzed to support or complement the survey data. We coded and extracted the transcript to get insights into the students’ learning experience, students’ understanding of the feature branch workflow, and overall impressions and motivations of collaborative programming.

6.4 Results

6.4.1 Survey Data

Table 6.2 summarizes the descriptive statistics (means and standard deviations) of the outcome variables measured in the three surveys. The statistical analysis using a linear mixed-effects model showed that use of scaffolded feature branch workflow is a statistically significant positive predictor of student engagement and perceived fairness (student engagement: t(52) = 2.297, p = 0.025, marginal R2 = .049, conditional R2 = .603, standardized regression coefficient: 0.28; perceived fairness: t(52) = 2.189, p = 0.033, marginal R2 = .028, conditional R2 = .654, standardized regression coefficient: 0.324) within the 95% confidence interval, getting aligned with the hypotheses. The treatment had a marginally significant positive effect on psychological ownership (t(52) = 1.984, p = 0.052, marginal R2 = .078, conditional R2 = .344, standardized regression coefficient: 0.439) within

Table 6.2: Descriptive statistics (mean and standard deviation)

Outcome Variable	M1 (SD1)	M2 (SD2)	M3 (SD3)
Student Engagement	5.4 (0.6679)	5.73 (0.7232)	5.73 (0.6838)
Psychological Ownership	4.59 (1.2171)	5.33 (1.08)	5.19 (1.0338)
Perceived Fairness	5.12 (1.2449)	5.56 (0.8485)	5.34 (1.0297)

the 90% confidence interval. The treatment appeared to have medium effect sizes on all three of the dependent variables as measured by Cohen's d [9] (student engagement: 0.553, psychological ownership: 0.641, and perceived fairness: 0.406). Figure 6.2 illustrates the treatment effects on student engagement, psychological ownership, and perceived fairness with and without the treatment.

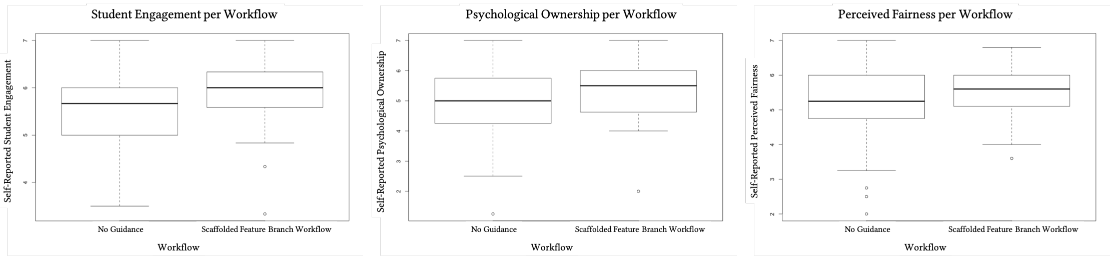


Figure 6.2: Effects of using scaffolded feature branch workflow on student engagement, psychological ownership, and perceived fairness

For those dependent variables where our treatment had significant effects on, we examined the effects of time to see if the effects were merely the outcome of repetition of projects over time. Time was also a significant predictor of student engagement (t(52) = 2.156, p = 0.035, standardized regression coefficient: 0.15) and psychological ownership (t(52) = 2.351, p = 0.023, standardized regression coefficient: 0.3) within the 95% confidence interval, but not of perceived fairness (t(52) = 1.278, p = 0.206, standardized regression coefficient:

0.11). This result indicates that the repetition of project over time wasn't a factor for the increase of perceived fairness, while it helped the increased levels of student engagement and psychological ownership. Figure 6.3 depicts the patterns of student engagement and psychological ownership over time throughout the intervention.

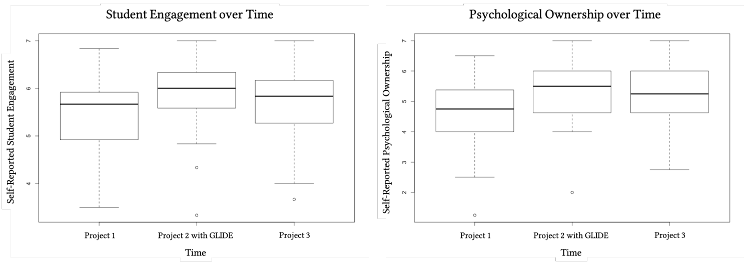


Figure 6.3: Effects of time on student engagement and psychological ownership

Figure 6.3 illustrates the increasing patterns of student engagement and psychological ownership from project 1 (with no treatment given) and project 2 (with the treatment given) and the subtle decreasing patterns of them between project 2 (with the treatment given) and project 3 (with the treatment taken away). We conducted a follow-up statistical analysis to examine whether the difference between project 2 and project 3 is significant. The two-tailed t-test between the two data points showed that there are no significant differences in both variables between project 2 and project 3 (student engagement: t(52) = 0.000, p = 1; psychological ownership: t(52) = 0.4866, p = 0.6286). This result indicates that the increased levels of student engagement and psychological ownership between project 1 and project 2 have sustained the levels without making notable drops.

6.4.2 Focus Group Interview Data

The participants commonly acknowledged the salient differences between the different ways of working together they went through: unstructured collaboration using Dreamweaver and following the feature branch workflow on GLIDE. When asked about the favorite features of the feature branch workflow and GLIDE also talked about the benefits of using separate branches in a shared repository for a group in exchanging information. From their quote, we were able to see their understanding on the core idea of feature branch workflow, which is individuals make their own contribution on branches, or separate logical copies.

> Arther: “In working together and viewing other people’s branch, I really like the feature when you put everything together in the master and you make a branch, then you have a copy of everything already, so you can work on that and you update it. Merging everything together is very simple.”

> Alice: “If you want to see someone else’s work in your team, you could just look into their branches, instead of going to their computer or copy their code.”

We also identified the students’ appreciation of and motivation for collaborative programming practice. Baker, for example, expressed his desire to learn more about GitHub responding to the question of how was the instructions on a collaborative workflow using GitHub and GLIDE. Furthermore, Bill, one of the most advanced students in this classroom, shared his anecdote on self-directed learning on Git command line tools after his project 2 using GLIDE, expressing his passion in learning collaborative programming skills as the next step beyond this class activities.

> Baker: “We didn’t actually use GitHub.We used GLIDE that’s working with GitHub. Let’s say we’re gonna graduate from this program and we go to work in September. We still wouldn’t know how to use GitHub. We only talked about that one, really small part [operations of creating repository and merging branches, which students performed on GitHub]. We know that one specific part. We don’t really understand GitHub overall. In the account section, there are more features we didn’t cover. [When he was asked if he wants to learn more about Git or GitHub] Yes, because that’s what developers use.”

> Bill: “There was a learning the Git command line for me, on my own, after this. [When he was asked what motivated his self-directed learning] I didn’t feel like I want to use GLIDE later, when I become a professional, it’s just awkward.”

6.5 Discussion

The results yielded several notable implications. First, students were able to use GLIDE to use feature branch workflow to complete collaborative website development projects. Second, the linear mixed-effects model suggests that our approach to collaborative projects using the scaffolded feature branch workflow and GLIDE turns out to be predictors of student engagement, perceived fairness in contribution and acknowledgement, and psychological ownership in the project, in line with the hypotheses. This result resonates with the significance of scaffolding [132], or guided participation [127], as emphasized in education literature. Due to the complexity of collaborative software development, learners need scaffolding to guide the process. Furthermore, the help at the adequate level, or the way that GLIDE scaffolded the Git workflow in this case, reflects the tools, norms, terminology, and the way of working, which helps the learning better contextualized in the long term. As mentioned in the related work section, this differentiates our unique approach from other collaborative programming platforms that implement unstructured forms of collaboration.

Another notable point in the result is that the treatment had medium effect size on all of those outcome variables. As reported in effect size by Cohen's d measurement, the treatment brought about positive shifts of the outcome distributions by approximately 0.4 0.6 times of the overall standard deviation in average. Considering that the effect size typically accepted in experimental studies in psychology is around Cohen's d = 0.4, the effect sizes of the treatment this research measured hover the general standard, indicating that the results of the statistical analysis don't only mean the significance of the estimated model for those three outcome variables, but also argues the treatment had the substantial size of effects that is empirically observable.

The interpretation of the effects of time (Figure 6.3) renders another set of noteworthy takeaways, as intended by the experimental design. First, the patterns of the time-based effects on the two outcome variables support our argument that the increase in outcome variables between project 1 and 2 was due to the treatment, rather than the effect of repeated project performance; the scores of the two outcome variables decrease once the treatment was removed. This means that the pure time-based effects were not enough to increase or decrease the levels of the outcome variables when the treatment was taken away (between project 2 and 3), whereas the positive effects were observed when it was provided (between project 1 and 2). Second, the levels of outcome variables in project 3 still stayed higher than those in project 1 based on the follow-up analysis on the time-based effects between project 2 and project 3, even though the scaffolding tool has been removed. We argue that the differences between those two separate measurements represent the learning gains through the intervention of this research; having learned and applied the abstract model of collaboration to their class activities, even without the software tool, may improve their learning experience in collaborative projects. This is an encouraging implication getting in line with our premise that the way of collaboration is a skill that can be developed and transferred to other contexts, instead of being tied to a specific software tool or a context.

This piece of finding suggests the most significant contribution of this work: the repeated tasks over time alone can't help implementing more equitable collaboration, while a special treatment, such as GLIDE, to reform the collaboration process can. One might think that young students are simply too premature to learn those abstract strategy for fair and effective collaboration, as Floyd stated in section 4.6.3. From this idea, it could be logical to conclude that when a student develops a skill to have the bigger picture about relevant tasks, such effective collaboration through fairer coordination and contribution can be achievable. However, the finding of this work states that GLIDE approach may give a way to actively facilitate the effective and fair collaboration process without leaving this goal onto mere repetition over time.

Considering that CS students in advanced levels and professionals in the field of software engineering in general learn Git command line tools and workflows at some point of their learning trajectories or careers, a scaffolding tool for more equitable collaboration process for younger students, such as GLIDE, generates a chance for early exposure to the fair collaboration workflow. This early exposure to the fair collaboration workflow implies that other techniques including other programming languages the young students will learn can build on top of the mental model for the collaborative process that they constructed while using GLIDE. Thus, this implication opens the possibility of further collaborative learning in other diverse subjects in CS beyond introductory web programming.

Deriving from the focus group, the students had their own interpretation of the collaborative workflow, which implies they didn't simply follow each step by rote as GLIDE scaffolds the process. Also, they were able to talk about their experiences using the terminology they learned in the six sessions of lecture during the intervention, such as repository, branch, and merge. This also supports the argument that our intervention was effective enough to engage them in active learning in collaborative workflow using Git. We think the quote from Baker in results section sums our hope that students should experience how effectively collaborate in a classroom and be able to make connections to what will happen beyond classrooms. Moreover, Bill's quote in the results section summarizes the value of GLIDE approach. If a scaffolding software merely provides a usability shortcut to a desired outcome, it is hard to claim the learning effects. However, GLIDE exposed the novice users to the learning in a new domain when they're ready to move on to the next step. These results contrast with the general perception that Git is hard and teaching young students how to use it is even harder. This implication suggests the significant potentials that CS education can build on top of the foundation of collaborative workflow, given proper ways of introducing the required techniques, knowledge, and adequate tools, which may foster more inclusive education environment and better reflect software engineering industry.

6.6 Summary

We designed and implemented a set of lesson plans and a supporting software tool, GLIDE, to teach novice students how to effectively work together in building websites in a scaffolded way using the feature branch workflow within Git. The field experiment showed that the approach is feasible, may contribute to a better learning experience, based on survey responses. The focus group also showed that the lesson plans had educational value and effects.

This research has several limitations. Overall, we could have done a more rigorous evaluation with a between-subjects approach or a larger sample. We tried to mitigate this issue as much by having an A-B-A within-subject experimental design, with repeated measurements, and mixed-effects model for analysis that takes multiple observations into account.

CHAPTER 7

GENERAL DISCUSSION AND CONCLUSION

This chapter provides points of discussion derived from the previous chapters' research approach as well as contribution and limitation. One of the contributions of this work is that we implemented social learning platform in online and physical education environment where students learn from software-guided interaction (MOOCchat and GLIDE). Another primary contribution of this work is to expand the common knowledge on how to properly teach collaborative programming in the field of CS education. Effective teaching and learning methods for collaborative programming have been an unanswered question in an under-explored research area. Our own data collection to explore teachers' practical requirement also identified such needs (chapter 4). In our approach to teach novice students to collaborate through the feature branch workflow, we found that the scaffolded workflow on Git version control system is effective in guiding students' lab activities. The following sections discuss what benefits can potentially be achieved by employing our approaches and what theoretical implications they have.

7.1 Benefits

7.1.1 Practical Contributions of MOOCchat

The most basic benefit of our first empirical approach, MOOCchat, is that it helps educators replicate the small group oriented teaching practice in online learning platform. In reality, teaching or learning is hardly an individual activity taking place in a classroom only for an individual student. Compared to this, online learning environment assumes an individual student with isolation from the others for the sake of convenience and for physical limitations. Even though the MOOCs platforms have had discussion boards for questions and answers, the absence of small group context has been one of the most easily noticeable limitations in such online settings. As suggested in existing literature, the context of small groups in learning in physical classrooms is a positive factor for effective learning [129, 128, 130]. In particular, Webb [130] found that it is a desirable teaching practice to structure group activity to require students to share their own explanations in linguistic forms with others. The direct benefit from MOOCchat in this regard is that students learning in online environments were able to learn in small groups matched online. They had an opportunity to explain the coursework materials (quiz on software engineering) in linguistic forms, similarly to Webb [130]'s work, to support their learning in the context of small groups. As little, if not none, about how to support the real-time interaction among students has been known when this work implemented MOOCchat approach, this work presented the initial experience of how teachers and students online could benefit from interactive learning in the online small group context.

Another benefit of MOOCchat approach is that it might help alleviate the attrition issue on large scale learning platforms. Online learning largely relies on an individual student's self-directed learning. In such isolated environment, the lack of interactivity has been pointed as the crucial factor for the attrition problem [87]. Also, supporting high quality interaction among the learners reportedly contributes to successful online learning [62, 63]. The bus terminal model for the matching process of MOOCchat approach made the real-time student interaction feasible, which in turn might contribute to better retention and completion of learning in the course, as the qualitative feedback from the participants in chapter 3 support.

7.1.2 Practical Contributions of Exploratory Study on Instructional Designs

The interview study with the secondary school teachers provided practical ideas about where and how teachers can try to improve instructional approaches in secondary CS education practice focused on web programming. We encourage teachers to use the instructional design – task and type of collaboration – task matrices as a guideline in planning the lessons in lab activity-centric classes; what knowledge to teach, what task to have the students practice, and what instructional designs or collaborative approaches to use to support them. The three propositions derived from the findings in section 4.6 also yield practical contribution that helps teachers interested in how to better teach web programming classes. We hope this work can directly inform the design, validation, and evaluation of instructional designs to help teachers and students seeking to teach or learn web programming in collaborative ways.

7.1.3 Practical Contributions of GLIDE

The empirical approach with GLIDE for collaborative programming in this work also illuminated the potential benefits. First, this work open-sourced[1] the

[1] Available at https://github.com/stlim0730/glide

educational software application we built, GLIDE[2]. In chapter 4, we specified the purpose the software is supposed to serve, which is to integrate scaffolded collaborative workflow in web programming courses for novice students. Since it is designed to satisfy the common needs discovered from teachers' report on their teaching methods, we argue that the audience of this work, who tries to teach and learn how to support the feature branch workflow in a scaffolded and novice-friendly way, can benefit from using GLIDE in teaching them how to collaboratively program.

The second practical contribution of GLIDE approach is the teaching practice we presented and the design we carried out; this includes the lesson plans (chapter 5), the educational context we designed (chapter 5), and students' learning experience as the expected results in the future (chapter 6). This work started from the problem statement of "learning how to collaborate has been left out of CS curriculum despite the significance of collaboration in its lab activities." As we presented how to effectively teach young students without deep background in CS, educators trying to design collaboration-oriented curricula can directly benefit from the lesson plans example to begin with. Furthermore, this work verified the feasibility of the lesson plans in an appropriate field-site, rather than leaving them to be tested. We call for the consideration of the difference between the educational context described in this work and the actual environment the audience might want to replicate this teaching practice.

In addition to educators, the students can also benefit from GLIDE approach. The collaborative workflow they can learn through this novel approach way of performing is not strictly tied to specific techniques, such as web programming, they use to build software. Even though this work focused on the introductory

[2] Available at https://glide.site

web programming course due to the several advantages for the procedure of the research, which include the engaging topic for novice and easy representation of the student project outcomes, the feature branch workflow is flexible enough to apply for virtually any types of software development tasks. Especially, the lesson plans and group project had a strong emphasis on the abstract mental model of how the collaboration process should be, rather than how to use a specific set of tools of Git. We expect that the students be able to generalize the general idea of the effective way of collaboration process to other software programming tasks throughout their further learning trajectory in CS.

7.2 Theoretical Implications

In this section, we try to reexamine the connection between our empirical approach and the learning theory on which this work is based. We will first discuss how social development theory backs up the learning on MOOCchat and GLIDE.

7.2.1 Revisiting MOOCchat and GLIDE: Focusing on Social Development Theory

Social development theory by Vygotsky [127] influenced this work as the fundamental framework of how learning occurs and how to facilitate it. The key takeaways of the theory around which this work built were 1) “guided participation”, also known as scaffolding, that helps learning in the zone of proximal development and 2) the two crucial roles of language, sign systems, and tools that mediate interaction with more knowledgeable others and students internalize learning materials, and 3) indirect interaction with the broader society through language as a socio-cultural product that better better contextualizes learning. This section reflects on the two empirical approaches we made regarding these three aspects.

Relying on the theoretical implications of the social development theory, the MOOCchat was an attempt to introduce student interaction into extremely decontextualized education settings. From the perspectives of social development theory, MOOCs open more knowledgeable others and interaction with them. On the contrary, the current work developed MOOCchat approach real-time matching feature to form ad-hoc groups of students, which gives them the chance to interact with the more knowledgeable others.

The way the students in a small group on MOOCchat interacted each other implements the ideas of interaction as the source of learning and language as the mediator for learning. The specific element intended for learning on MOOCchat is where the students exchange their ideas in the discussion and have a chance to change their answers to finally submit. During the discussion, the real-time chat induces them to reason their initial answers, persuade others, identify errors from others' references, and negotiate on the final answers, which in turn help the active process of internalization where each student uses one's own language for this whole procedure, rather than passively copying others' references. Thus, we argue that MOOCchat enables student interaction for learning and also structures their interaction for effective internalization through active processing of language.

Learning in using GLIDE was also founded on social development theory.

First, the design of GLIDE is rooted in guided participation. The feature branch workflow is generally considered as a difficult topic as reported in the interviews with the teachers. The tasks required to perform a project using the feature branch workflow are out of the students' reach, considering their actual developmental level and potential developmental level. Therefore, to pull those tasks down to the area within the zone of proximal development, selected tasks among the hard ones had to be scaffolded; using command line interface, having many options per command, and understanding the abstract model of Git workflows. To achieve this, GLIDE was designed as a web-based application with the graphical user interface instead of command line, dedicated to support one simple target workflow (the feature branch workflow) while sacrificing the high levels of flexibility with the commands and options, and tightly coupled with the lecture to help building the abstract model of the workflow before working on software development. This is where the participating teacher's advice was critical in finding the appropriate levels of difficulty of the tasks, to include them in the common zone of proximal development for the students in the class.

The second way that learning in GLIDE approach reflected social development theory is that the lesson plans for the lecture placed a significant focus on being able to explain the feature branch workflow to others and discuss it. The lesson plans had the dedicated time for peer discussion where the students ask, answer, and explain what they understand in their own language. This reflects the mediation role of language during the interaction with the more knowledgeable others in the theory and the findings from the qualitative analysis suggested that it might have worked as the source of learning. The ideas exchanged are also supposed to be internalized in language form, which is the key step of learning according to the theory.

Lastly, GLIDE approach also put significant focus on giving the students the exposure to the standard terminology and widely accepted norms (e.g., not writing code off directly on the master branch and keeping the commit tree as simple as possible) as implicit sign systems and language from the culture of collaborative programming in industry practice. Social development theory explains the value of this teaching practice that the learners can indirectly interact with the broader society through the mediating language. By teaching with this concentration, we argue that learning on GLIDE can be better contextualized in the society of collaborative programming, which potentially reduces the gap between learning context and the practice in industry or academia.

7.2.2 Theoretical Contributions of Exploratory Study on Instructional Designs

The theoretical contribution of the interview study includes a discussion of how instructional designs from cognitive load theory and cognitive apprenticeship have been contextualized in web programming tasks. We also illustrated how those theoretical constructs interplay with the task types that students perform and collaborative approaches. Another theoretical contribution is that this work proposed the instructional design and collaboration task matrix as an analytical framework that other researchers can use to understand and explore the interaction between these factors.

7.3 Limitations and Challenges

Despite the contributions and implications of this work discussed above, this work has several limitations.

First, we were not able to have the large sample sizes for quantitative analyses in MOOCchat and GLIDE approaches. Larger sample sizes might have presented more convincing results on the learning gains through MOOCchat or the statistical estimates for the treatment effects of GLIDE. Our approach to minimize the latter issue was to take the linear mixed-effects model that preserves observations for each data point, rather than losing by aggregating them.

Second, our observational study didn’t have multiple trials of fieldwork. This limited the chance of more diverse observation and the robustness of hypothesis testing. More diverse observation might have provided more fresh insights to the phenomena we observed. Furthermore, incremental variation in the treatment between trials might have generated interesting findings that we could not observe in this work. For example, after the quantitative evaluation on the students’ learning experience in chapter 6, another similar trial of the intervention in a comparable settings with a few user interface tweaks (e.g., showing commit tree for task awareness cue or other branches with user avatars for a group awareness cue) on the software would have yielded the chance to examine the influence of the changes made.

We argue that the two limitations of this work presented above stem from one practical challenge we experienced throughout the research procedure. The most challenging part of conducting fieldwork was to establish partnerships with schools. In this sort of research that engages education entities or local communities, establishing strong connections as stakeholders and partners for the research gets huge significance. However, it was practically hard to find partnerships available because of several reasons. The two most critical barriers we encountered through this dissertation work were 1) the policy of schools and online learning providers giving not enough flexibility in lessons to allow our intervention and 2) teachers' hesitation for participation due to the perceived workload for a joint work. To overcome the limitations above, we hope to see more attempts to bring novel approaches to better learning experience and outcomes through partnerships between education practitioners and scholars in CS education research community.

7.4 Future Work

This section sheds light on how this work can be expanded, based on the research procedure and the findings reported. By answering those questions proposed below, we believe that follow-up research can build on top of the two empirical approaches of MOOCchat and GLIDE.

7.4.1 How to Better Facilitate Student Discussion through Better Matching Schemes?

One of the two key features of MOOCchat was the real-time matching to form ad-hoc small groups of students in order to give them a change to have more knowledgeable others to teach or learn from. One of the limitations of the current version of the software reported in chapter 3 was that it didn't have any matching schemes to form the student groups. To better facilitate students discussion, which is the source of their learning on MOOCchat, future work may try various criteria to form better combinations of discussants. One potentially promising matching scheme is to make each small group have the maximum possible variety of initial answer submissions, so that they may exchange argument in the discussion and identify their logical fallacy. This might have the students more actively engage in the discussion and also contribute to fairer chance to learn on MOOCchat, because a group of students with the same initial submissions can't get the same quality of discussion as another group of students with more variety of initial choices; the different points of view shared in the discussion will make it richer.

7.4.2 How Will GLIDE Approach Help Learning Advanced Level Techniques?

GLIDE approach helped novice students learn the abstract model of collaboration process. Bill's anecdote on self-directed learning on Git command line tools after our intervention in chapter 6 suggests a potential follow-up question to ask: how and how much will the initial experience with GLIDE help learning the advanced Git techniques for collaboration? GLIDE approach gave the students early exposure to basic terminology and the feature branch workflow. Two of the reasons Git is widely considered hard are the barrier that the terminology (jargon) creates and the implicit workflow; for example, it is possible for a user to know what a branch means and the command line "git branch mybranch" does (with the help from references or by rote memory), but not what should come later for integration process. Based on the effectiveness of GLIDE approach in establishing the abstract model of collaboration process, a possible answer might be that students having the experience with GLIDE learn or perform better in learning or using the advanced level collaboration techniques.

Another future work related to the advanced level techniques is to equip GLIDE software with command line console that accepts Git commands, which is an optional advanced feature. This will provide the user with an opportunity to practice and learn the command line tools as well as the practical benefit of the flexibility of Git command line tools.

7.4.3 How Can Non-CS Majors Benefit from GLIDE Approach?

As the applicability and usefulness of programming increases in diverse disciplines, students in other areas who write computer code can benefit from the scaffolded collaborative programming that GLIDE approach introduced. The examples of programming code or script that non-CS majors write include code for statistical or scientific analysis in R or MATLAB, computational designs or architecture code in Rhino, and visual designs and arts in Processing. Similarly to the problem identification within CS discussed in chapter 1, education in these non-CS areas is also extremely syntax-oriented, while collaboration still has significant emphasis across the areas. This suggests another potential contribution that GLIDE approach can be employed to support learning through collaborative programming activities in those non-CS areas.

A few additional features in GLIDE software are required to support those programming languages because the current version we used for this work only supports the three web programming languages of HTML, CSS, and JavaScript. The two essential features to be required include the syntax highlighting of the editor for each programming language and the live preview adjusted to support rendered outputs or console outputs. Except for these two language-specific features, the workflow-related skills students can learn while using GLIDE remains the same, as well as the overall benefit of using the standard version control system.

GLIDE approach in this work has focused on lab activities for web programming, but the potential of Git as a flexible collaboration framework also apply to other types of digital content creation work. In other words, future work may extend this work to identifying effective collaboration workflows for digital content generation activities that are not related to coding (e.g., 3D modeling or image / audio editing) and how to better teach students the workflow. Git is a highly flexible tool that can keep track of changes of not only text files (e.g., source code), but binary files (e.g., images or audio), so that it can be used for several different kinds of projects. Non-coding content generation activities might be better supported by different workflows than the feature branch workflow because of the different ways of coordination and integration, which might require different iterative process. These remaining questions may expand learning how to collaborate.

7.5 Conclusion

Overall, the series of studies presented in this dissertation work has enriched our understanding of how to implement collaborative programming in CS education. Social development theory informed us about how learning occurs through student interaction and how to facilitate it. To facilitate online learning where student interaction has been largely missing, we found the real-time discussion on MOOCchat improving the users' quiz performance. And then, we explored what the current teachers want to teach, what needs to be taught, and how the instructions should be. From the exploration of the educational needs, we found the set of lesson plans to teach a standard collaboration workflow with the supporting software tool, GLIDE, made the novel teaching approach feasible and the student collaboration process more equitable.

As the primary findings of this work suggest, well-structured collaboration process may lead us to more equitable contribution among the students. As more and more students, including younger age groups and non-CS majors, will want to or be required to learn programming skills in modern society, CS classrooms are going to experience the increased levels of diversity across the levels. In addressing and embracing the differences in background and skill-sets between learners through collaborative programming education, we hope this work provides a potential groundwork.

BIBLIOGRAPHY

[1] Code.org.

[2] CS4all: NYC.

[3] D. Robert Adams. Integration early: a new approach to teaching web application development. *dl.acm.org*, 23(1):97–104, 2007.

[4] Eric Allen, Robert Cartwright, and Brian Stoler. DrJava. In *Proceedings of the 33rd SIGCSE technical symposium on Computer science education - SIGCSE '02*, volume 34, pages 137–137. ACM Press, 2002.

[5] William F. Atchison, Earl J. Schweppe, William Viavant, David M. Young, Samuel D. Conte, John W. Hamblen, Thomas E. Hull, Thomas A. Keenan, William B. Kehl, Edward J. McCluskey, Silvio O. Navarro, and Werner C. Rheinboldt. Curriculum 68: Recommendations for academic programs in computer science: a report of the ACM curriculum committee on computer science. *Communications of the ACM*, 11(3):151–197, March 1968.

[6] Atlassian. Git Feature Branch Workflow | Atlassian Git Tutorial.

[7] Atlassian. Git Workflow | Atlassian Git Tutorial.

[8] Atlassian. What is version control | Atlassian Git Tutorial.

[9] Richard H. Austing, Bruce H. Barnes, Della T. Bonnette, Gerald L. Engel, and Gordon Stokes. Curriculum '78: recommendations for the undergraduate program in computer science— a report of the ACM curriculum committee on computer science. *Communications of the ACM*, 22(3):147–166, March 1979.

[10] Sasha Barab and Kurt Squire. Design-Based Research: Putting a Stake in the Ground. *Journal of the Learning Sciences*, 13:1–14, January 2004.

[11] Sasha A. Barab and Jonathan A. Plucker. Smart People or Smart Contexts? Cognition, Ability, and Talent Development in an Age of Situated Approaches to Knowing and Learning. *Educational Psychologist*, 37(3):165–182, September 2002.

[12] Brigid Barron. When Smart Groups Fail. *Journal of the Learning Sciences*, 12(3):307–359, July 2003.

[13] Douglas Bates, Martin Mchler, Ben Bolker, and Steve Walker. Fitting Linear Mixed-Effects Models Using lme4. *Journal of Statistical Software*, 67(1):1–48, October 2015.

[14] Jon Beck. Fair Division As a Means of Apportioning Software Engineering Class Projects. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '08, pages 68–71, New York, NY, USA, 2008. ACM. event-place: Portland, OR, USA.

[15] Mordechai Ben-Ari. Constructivism in Computer Science Education. In *Proceedings of the Twenty-ninth SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '98, pages 257–261, New York, NY, USA, 1998. ACM.

[16] Mordechai Ben-Ari. Constructivism in Computer Science Education. *Journal of Computers in Mathematics and Science Teaching*, 20(1):45–73, 2001.

[17] Esmail Bonakdarian. Pushing Git & GitHub in Undergraduate Computer Science Classes. *J. Comput. Sci. Coll.*, 32(3):119–125, January 2017.

[18] Kristy Elizabeth Boyer, August A. Dwight, R. Taylor Fondren, Mladen A. Vouk, and James C. Lester. A Development Environment for Distributed Synchronous Collaborative Programming. In *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '08, pages 158–162, New York, NY, USA, 2008. ACM.

[19] John Britton and Tim Berglund. Using Version Control in the Classroom (Abstract Only). In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 753–753, New York, NY, USA, 2013. ACM. event-place: Denver, Colorado, USA.

[20] Darci Burdge and Stoney Jackson. Git 101: Foundations for a Common Workflow to Contribute to HFOSS: Tutorial Presentation. *J. Comput. Sci. Coll.*, 31(6):18–20, June 2016.

[21] Darci Burdge and Stoney Jackson. Git 102: A Common Workflow to Contribute to HFOSS: Tutorial Presentation. *J. Comput. Sci. Coll.*, 31(6):34–36, June 2016.

[22] Michael E. Caspersen and Jens Bennedsen. Instructional design of a programming course. In *Proceedings of the third international workshop on Computing education research - ICER '07*, pages 111–111. ACM Press, 2007.

[23] Lillian Cassel, Alan Clements, Gordon Davies, Mark Guzdial, Rene McCauley, Andrew McGettrick, Bob Sloan, Larry Snyder, Paul Tymann, and Bruce W. Weide. Computer Science Curriculum 2008: An Interim Revision of CS 2001. Technical report, ACM, New York, NY, USA, 2008.

[24] Paul Chandler and John Sweller. Cognitive Load Theory and the Format of Instruction. *Cognition and Instruction*, 8(4):293–332, 1991.

[25] Carl Chang, Peter J. Denning, James H. Cross II, Gerald Engel, Robert Sloan, Doris Carver, Richard Eckhouse, Willis King, Francis Lau, Susan Mengel, Pradip Srimani, Eric Roberts, Russell Shackelford, Richard Austing, C. Fay Cover, Gordon Davies, Andrew McGettrick, G. Michael Schneider, and Ursula Wolz. Computing Curricula 2001: Final Report. Technical report, 2001.

[26] Joe D. Chase and Edward G. Okie. Combining cooperative learning and peer instruction in introductory computer science. In *ACM SIGCSE Bulletin*, volume 32, pages 372–376. ACM, 2000.

[27] Ed H. Chi and Ruben Ortega. Expanding CS Education; Improving Software Development. *Commun. ACM*, 53(9):8–9, September 2010.

[28] Herbert H. Clark, Robert Schreuder, and Samuel Buttrick. Common ground at the understanding of demonstrative reference. *Journal of verbal learning and verbal behavior*, 22(2):245–258, 1983.

[29] Peter J. Clarke, Debra Davis, Tariq M. King, Jairo Pava, and Edward L. Jones. Integrating Testing into Software Engineering Courses Supported by a Collaborative Learning Environment. *Trans. Comput. Educ.*, 14(3):18:1–18:33, October 2014.

[30] P Cobb, K McClain, and K Gravemeijer. Learning about statistical covariation. *Cognition and instruction*, 2003.

[31] Derrick Coetzee, Seongtaek Lim, Armando Fox, Bjorn Hartmann, and Marti A. Hearst. Structuring interactions for large-scale synchronous peer learning. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, pages 1139–1152. ACM, 2015.

[32] Allan Collins, John Seely Brown, and Ann Holum. Cognitive apprenticeship: Making thinking visible. *American Educator*, 15:6–11, 38–46, 1991.

[33] Allan Collins, John Seely Brown, and Susan E Newman. Cognitive apprenticeship: teaching the craft of reading, writing, and mathtematics. *Center for the Study of Reading Technical Report; no. 403.*, 1987.

[34] Catherine H. Crouch and Eric Mazur. Peer Instruction: Ten years of experience and results. *American Journal of Physics*, 69(9):970–977, August 2001.

[35] Sayamindu Dasgupta, William Hale, Andrs Monroy-Hernndez, and Benjamin Mako Hill. Remixing as a Pathway to Computational Thinking. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing - CSCW '16*, pages 1436–1447. ACM Press, 2016.

[36] Richard Davis, Yasmin Kafai, Veena Vasudevan, and Eunkyoung Lee. The Education Arcade: Crafting, Remixing, and Playing with Controllers for Scratch Games. In *Proceedings of the 12th International Conference on Interaction Design and Children*, IDC '13, pages 439–442, New York, NY, USA, 2013. ACM.

[37] Louis Deslauriers, Ellen Schelew, and Carl Wieman. Improved learning in a large-enrollment physics class. *science*, 332(6031):862–864, 2011.

[38] R. Duque and C. Bravo. Analyzing Work Productivity and Program Quality in Collaborative Programming. In *2008 The Third International Conference on Software Engineering Advances*, pages 270–276, October 2008.

[39] Lisa Ede and Andrea Lunsford. *Singular Texts/Plural Authors: Perspectives on Collaborative Authoring*. Southern Illinois University Press, Carbondale, 1990.

[40] Stephen H. Edwards, Daniel S. Tilden, and Anthony Allevato. Pythy. In *Proceedings of the 45th ACM technical symposium on Computer science education - SIGCSE '14*, pages 641–646. ACM Press, 2014.

[41] Gijsbert Erkens, Jos Jaspers, Maaike Prangsma, and Gellof Kanselaar. Coordination Processes in Computer Supported Collaborative Writing. *Comput. Hum. Behav.*, 21(3):463–486, May 2005.

[42] Martin R. Fellenz. Toward Fairness in Assessing Student Groupwork: A Protocol for Peer Evaluation of Individual Contributions. *Journal of Management Education*, 30(4):570–591, August 2006.

[43] Deborah A. Fields, Veena Vasudevan, and Yasmin B. Kafai. The Programmers' Collective: Connecting Collaboration and Computation in a High School Scratch Mashup Coding Workshop. June 2014.

[44] K-12 Computer Science Framework. K-12 Computer Science Framework.

[45] Jos Fransen, Paul A. Kirschner, and Gijsbert Erkens. Mediating team effectiveness in the context of collaborative learning: The importance of team and task awareness. *Computers in Human Behavior*, 27(3):1103 – 1113, 2011.

[46] William Frawley. *Vygotsky and Cognitive Science: Language and the Unification of the Social and Computational Mind*. Harvard University Press, 1997.

[47] Jeanne Marcum Gerlach. Is this collaboration? *New Directions for Teaching and Learning*, 1994(59):5–14, 1994.

[48] Catherine Wilson Gillespie and Sally Beisser. Developmentally Appropriate LOGO Computer Programming with Young Children. *Information Technology in Childhood Education Annual*, 2001(1):229–244, 2001.

[49] GitHub. GitHub Pages.

[50] Robert Glaser. Education and thinking: The role of knowledge. *American Psychologist*, 39(2):93–104, 1984.

[51] Tom Gross, Chris Stary, and Alex Totter. User-centered awareness in computer-supported cooperative work-systems: Structured embedding of findings from social sciences. *International Journal of Human-Computer Interaction*, 18(3):323–360, 2005.

[52] Philip J. Guo. Online python tutor. In *Proceeding of the 44th ACM technical symposium on Computer science education - SIGCSE '13*, pages 579–579. ACM Press, 2013.

[53] Said Hadjerrouit and Said. Java as first programming language. *ACM SIGCSE Bulletin*, 30(2):43–47, June 1998.

[54] Ken T. N. Hartness. Eclipse and CVS for Group Projects. *J. Comput. Sci. Coll.*, 21(4):217–222, April 2006.

[55] James E. Heliotis and James E. Easing up on the introductory computer science syllabus. In *Proceeding of the 24th ACM SIGPLAN conference com-*

panion on Object oriented programming systems languages and applications - OOPSLA '09, pages 683–683. ACM Press, 2009.

[56] Morten Hertzum. Collaborative information seeking: The combined activity of information seeking and collaborative grounding. *Information Processing & Management*, 44(2):957–962, March 2008.

[57] Christopher D. Hundhausen. Integrating Algorithm Visualization Technology into an Undergraduate Algorithms Course: Ethnographic Studies of a Social Constructivist Approach. *Comput. Educ.*, 39(3):237–260, November 2002.

[58] Teresa Hbscher-Younger and N. Hari Narayanan. Constructive and Collaborative Learning of Algorithms. In *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '03, pages 6–10, New York, NY, USA, 2003. ACM.

[59] Ville Isomttnen and Michael Cochez. Challenges and Confusions in Learning Version Control with Git. In *Information and Communication Technologies in Education, Research, and Industrial Applications Communications in Computer and Information Science: 10th International Conference, ICTERI 2014, Kherson, Ukraine, June 9-12, 2014*, pages 178–193. Springer, 2014.

[60] Maya Israel, Quentin M. Wherfel, Saadeddine Shehab, Oliver Melvin, and Todd Lash. Describing Elementary Students' Interactions in K-5 Puzzle-based Computer Science Environments Using the Collaborative Computing Observation Instrument (C-COI). In *Proceedings of the 2017 ACM Conference on International Computing Education Research*, ICER '17, pages 110–117, New York, NY, USA, 2017. ACM.

[61] Jeroen Janssen, Femke Kirschner, Gijsbert Erkens, Paul A. Kirschner, and Fred Paas. Making the Black Box of Collaborative Learning Transparent: Combining Process-Oriented and Cognitive Load Approaches. *Educational Psychology Review*, 22(2):139–154, June 2010.

[62] Hanan Khalil and Martin Ebner. Interaction possibilities in MOOCshow do they actually happen? In *3rd International Conference on Higher Education Development” Future Visions for Higher Education Development”*, pages 1–24. ., 2013.

[63] Hanan Khalil and Martin Ebner. MOOCs completion rates and possible methods to improve retention-A literature review. In *EdMedia+ Innovate*

Learning, pages 1305–1313. Association for the Advancement of Computing in Education (AACE), 2014.

[64] Femke Kirschner, Fred Paas, and Paul A. Kirschner. A Cognitive Load Approach to Collaborative Learning: United Brains for Complex Tasks. *Educational Psychology Review*, 21(1):31–42, March 2009.

[65] Timothy Koschmann. *CSCL : Theory and Practice of An Emerging Paradigm*. Routledge, October 2012.

[66] Oren Laadan, Jason Nieh, and Nicolas Viennot. Teaching Operating Systems Using Virtual Appliances and Distributed Version Control. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, SIGCSE '10, pages 480–484, New York, NY, USA, 2010. ACM. event-place: Milwaukee, Wisconsin, USA.

[67] Bruce E Larson. Classroom discussion: a method of instruction and a curriculum outcome. *Teaching and Teacher Education*, 16(5):661–677, July 2000.

[68] Joseph Lawrance, Seikyung Jung, and Charles Wiseman. Git on the cloud in the classroom. In *Proceeding of the 44th ACM technical symposium on Computer science education - SIGCSE '13*, pages 639–639. ACM Press, 2013.

[69] Arthur H. Lee. A manageable web software architecture. In *Proceedings of the 34th SIGCSE technical symposium on Computer science education - SIGCSE '03*, volume 35, pages 229–229. ACM Press, 2003.

[70] Carol D. Lee and Peter Smagorinsky. *Vygotskian perspectives on literacy research: Constructing meaning through collaborative inquiry*. Cambridge University Press, 2000.

[71] Cen Li, Zhijiang Dong, Roland H. Untch, and Michael Chasteen. Engaging computer science students through gamification in an online social network based collaborative learning environment. *International Journal of Information and Education Technology*, 3(1):72, 2013.

[72] Billy B. L. Lim. Teaching Web development technologies in CS/IS curricula. In *Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education - SIGCSE '98*, volume 30, pages 107–111. ACM Press, 1998.

[73] Seongtaek Lim, Rama Adithya Varanasi, and Tapan Parikh. GLIDE (Git-Learning IDE; Integrated Development Environment): In-class Collaboration in Web Engineering Curriculum for Youths (Abstract Only). In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, SIGCSE '18, pages 1112–1112, New York, NY, USA, 2018. ACM. event-place: Baltimore, Maryland, USA.

[74] Raymond Lister. After the gold rush: Toward sustainable scholarship in computing. In *Proceedings of the tenth conference on Australasian computing education (ACE '08)*, pages 3–17. Australian Computer Society, Inc., 2008.

[75] Thomas R. Lord. A comparison between traditional and constructivist teaching in college biology. *Innovative Higher Education*, 21(3):197–216, 1997.

[76] Thomas R. Lord. 101 reasons for using cooperative learning in biology teaching. *The American Biology Teacher*, 63(1):30–38, 2001.

[77] Brian Magerko, Jason Freeman, Tom McKlin, Scott McCoid, Tom Jenkins, and Elise Livingston. Tackling Engagement in Computing with Computational Music Remixing. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 657–662, New York, NY, USA, 2013. ACM.

[78] Qusay H. Mahmoud. A mobile web-based approach to introductory programming. In *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education - ITiCSE '11*, pages 334–334. ACM Press, 2011.

[79] Robert McCormick. Conceptual and Procedural Knowledge. *International Journal of Technology and Design Education*, 7(1-2):141–159, January 1997.

[80] Rebecca Mercuri, Nira Herrmann, and Jeffrey Popyack. Using HTML and JavaScript in introductory programming courses. In *Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education - SIGCSE '98*, volume 30, pages 176–180. ACM Press, 1998.

[81] Bertrand Meyer. Beyond Folk Pedagogy, 2018.

[82] Matthew B Miles and A Michael Huberman. Qualitative data analysis: An expanded sourcebook. 1994. *Beverly Hills: Sage Publications*, 1994.

[83] Barbara J. Millis. *Cooperative learning in higher education: Across the disciplines, across the academy*. Stylus, 2010.

[84] Hiroshi Natsu, Jesus Favela, Alberto L. Morn, Dominique Decouchant, and Ana M. Martinez-Enriquez. Distributed Pair Programming on the Web. In *Proceedings of the 4th Mexican International Conference on Computer Science*, ENC '03, pages 81–, Washington, DC, USA, 2003. IEEE Computer Society.

[85] Allen Newell, Paul S Rosenbloom, and John E Laird. Symbolic Architectures for Cognition. 1989.

[86] John Nordlof. Vygotsky, Scaffolding, and the Role of Theory in Writing Center Work. *The Writing Center Journal*, 34(1):45–64, 2014.

[87] Rena M. Palloff and Keith Pratt. *The virtual student: A profile and guide to working with online learners*. John Wiley & Sons, 2003.

[88] Elizabeth Patitsas, Michelle Craig, and Steve Easterbrook. Comparing and contrasting different algorithms leads to increased student learning. In *Proceedings of the ninth annual international ACM conference on International computing education research - ICER '13*, pages 145–145. ACM Press, 2013.

[89] Michael Quinn Patton. *Qualitative evaluation and research methods*. SAGE Publications, inc, 1990.

[90] Jean Piaget and Margaret Cook. *The origins of intelligence in children*, volume 8. International Universities Press New York, 1952.

[91] Jules M. Pieters and Henneke F. M. de Bruijn. Learning Environments for Cognitive Apprenticeship: From Experience to Expertise. In *Cognitive Tools for Learning*, pages 241–248. Springer Berlin Heidelberg, Berlin, Heidelberg, 1992.

[92] Yizhou Qian and James Lehman. Students' Misconceptions and Other Difficulties in Introductory Programming. *ACM Transactions on Computing Education*, 18(1):1–24, October 2017.

[93] Atanas Radenski. "Python first". In *Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education - ITICSE '06*, volume 38, pages 197–201. ACM Press, 2006.

[94] Anthony Ralston and Mary Shaw. Curriculum ’78—is computer science really that unmathematical? *Communications of the ACM*, 23(2):67–70, February 1980.

[95] Matt Ratto, R. Benjamin Shapiro, Tan Minh Truong, and William G. Griswold. The activeclass project: Experiments in encouraging classroom participation. In *Designing for Change in Networked Learning Environments*, pages 477–486. Springer, 2003.

[96] David Reed. Rethinking CS0 with JavaScript. *ACM SIGCSE Bulletin*, 33(1):100–104, March 2001.

[97] Karen L. Reid and Gregory V. Wilson. Learning by Doing: Introducing Version Control As a Way to Manage Student Assignments. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE ’05, pages 272–276, New York, NY, USA, 2005. ACM. event-place: St. Louis, Missouri, USA.

[98] Alexander Renkl, Robin Stark, Hans Gruber, and Heinz Mandl. Learning from Worked-Out Examples: The Effects of Example Variability and Elicited Self-Explanations. *Contemporary Educational Psychology*, 23(1):90–108, 1998.

[99] Mitchel Resnick, Brian Silverman, Yasmin Kafai, John Maloney, Andrs Monroy-Hernndez, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, and Jay Silver. Scratch: Programming for All. *Communications of the ACM*, 52(11):60–67, November 2009.

[100] Jochen Rick and Mark Guzdial. Situating CoWeb: a scholarship of application. *International Journal of Computer-Supported Collaborative Learning*, 1(1):89–115, March 2006.

[101] Debra Sanders and Dorette Sugg Welk. Strategies to scaffold student learning: applying Vygotsky’s Zone of Proximal Development. *Nurse educator*, 30(5):203–207, 2005.

[102] Luis Miguel Serrano Cmara, Maximiliano Paredes Velasco, and Jess ngel Velzquez-Iturbide. Evaluation of a Collaborative Instructional Framework for Programming Learning. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE ’12, pages 162–167, New York, NY, USA, 2012. ACM.

[103] Russell Shackelford, Barry Lunt, Andrew McGettrick, Robert Sloan, Heikki Topi, Gordon Davies, Reza Kamali, James Cross, John Impagliazzo, Richard LeBlanc, Russell Shackelford, Andrew McGettrick, Robert Sloan, Heikki Topi, Gordon Davies, Reza Kamali, James Cross, John Impagliazzo, Richard LeBlanc, and Barry Lunt. Computing Curricula 2005: The Overview Report. Technical Report 1595932593, ACM Press, New York, New York, USA, 2005.

[104] N Shah, C Lewis, and Roxane Caires. Analyzing equity in collaborative learning situations: A comparative case study in elementary computer science. In *Proceedings of International Conference of the Learning Sciences, ICLS*, volume 1, pages 495–502, January 2014.

[105] Niral Shah, Colleen M. Lewis, Roxane Caires, Nasar Khan, Amirah Qureshi, Danielle Ehsanipour, and Noopur Gupta. Building Equitable Computer Science Classrooms: Elements of a Teaching Approach. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 263–268, New York, NY, USA, 2013. ACM. event-place: Denver, Colorado, USA.

[106] David Shernoff, Mihaly Csikszentmihalyi, Barbara Shneider, and Elisa Shernoff. Student Engagement in High School Classrooms from the Perspective of Flow Theory. *School Psychology Quarterly*, 18(2):158–176, 2003.

[107] David Shernoff and Lisa Hoogstra. Continuing motivation beyond the high school classroom. *New Directions for Child and Adolescent Development*, 2001(93):73–87, 2001.

[108] Beth Simon, Ruth Anderson, Crystal Hoyer, and Jonathan Su. Preliminary Experiences with a Tablet PC Based System to Support Active Learning in Computer Science Courses. In *Proceedings of the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, ITiCSE '04, pages 213–217, New York, NY, USA, 2004. ACM.

[109] Barbara Leigh Smith and Jean T. MacGregor. *What is collaborative learning*. Washington, 1992.

[110] Michelle K. Smith, William B. Wood, Wendy K. Adams, Carl Wieman, Jennifer K. Knight, Nancy Guild, and Tin Tin Su. Why peer discussion improves student performance on in-class concept questions. *Science*, 323(5910):122–124, 2009.

[111] Leonard Springer, Mary Elizabeth Stanne, and Samuel S. Donovan. Ef-

fects of Small-Group Learning on Undergraduates in Science, Mathematics, Engineering, and Technology: A Meta-Analysis. *Review of Educational Research*, 69(1):21–51, March 1999.

[112] Maitta Spronken, Rob W. Holland, Bernd Figner, and Ap Dijksterhuis. Temporal focus, temporal distance, and mind-wandering valence: Results from an experience sampling and an experimental study. *Consciousness and Cognition*, 41:104–118, April 2016.

[113] Gerry Stahl, Timothy Koschmann, and Dan Suthers. Computer-supported collaborative learning: An historical perspective. In *Cambridge handbook of the learning sciences*. Cambridge University Press, Cambridge, UK, February 2014.

[114] Chris Stephenson and Cameron Wilson. Reforming K-12 computer science education what will your story be? *ACM Inroads*, 3(2):43–46, 2012.

[115] Marty Stepp, Jessica Miller, and Victoria Kirst. A "CS 1.5" introduction to web programming. In *Proceedings of the 40th ACM technical symposium on Computer science education - SIGCSE '09*, volume 41, pages 121–121. ACM Press, 2009.

[116] Anselm Strauss and Juliet M Corbin. *Basics of qualitative research: Grounded theory procedures and techniques.* Sage Publications, Inc, 1990.

[117] R Core Team. R: a language and environment for statistical computing, 2018.

[118] Despina Tsompanoudi, Maya Satratzemi, and Stelios Xinogalos. Exploring the Effects of Collaboration Scripts Embedded in a Distributed Pair Programming System. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '13, pages 225–230, New York, NY, USA, 2013. ACM.

[119] Allen B. Tucker and Allen B. Computing Curricula 1991. *Communications of the ACM*, 34(6):68–84, June 1991.

[120] Tomoyuki Urai, Takeshi Umezawa, and Noritaka Osawa. Enhancements to Support Functions of Distributed Pair Programming Based on Action Analysis. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '15, pages 177–182, New York, NY, USA, 2015. ACM.

[121] Linn Van Dyne and Jon L. Pierce. Psychological ownership and feelings of possession: Three field studies predicting employee attitudes and organizational citizenship behavior. *Journal of Organizational Behavior*, 25(4):439–459, 2004.

[122] Jeroen J. G. Van Merrinboer. Strategies for Programming Instruction in High School: Program Completion vs. Program Generation. *Journal of Educational Computing Research*, 6(3):265–285, August 1990.

[123] D. Van Strien. An Introduction to Version Control Using GitHub Desktop. *The Programming Historian*, June 2016.

[124] Aurora Vizcano, Juan Contreras, Jess Favela, and Manuel Prieto. An Adaptive, Collaborative Environment to Develop Good Habits in Programming. In Gilles Gauthier, Claude Frasson, and Kurt VanLehn, editors, *Intelligent Tutoring Systems*, Lecture Notes in Computer Science, pages 262–271. Springer Berlin Heidelberg, 2000.

[125] Lev S. Vygotsky. *Language and thought*. Massachusetts Institute of Technology Press, Ontario, Canada, 1962.

[126] Lev S. Vygotsky. Interaction between learning and development. *Readings on the development of children*, 23(3):34–41, 1978.

[127] Lev S. Vygotsky. *Mind in society: The development of higher psychological processes*. Harvard university press, 1980.

[128] Noreen M. Webb. Peer interaction and learning in cooperative small groups. *Journal of Educational Psychology*, 74(5):642–655, 1982.

[129] Noreen M. Webb. Peer interaction and learning in small groups. *International Journal of Educational Research*, 13(1):21–39, January 1989.

[130] Noreen M. Webb. Task-Related Verbal Interaction and Mathematics Learning in Small Groups. *Journal for Research in Mathematics Education*, 22(5):366, November 1991.

[131] Jacqueline Whalley and Nadia Kasto. A Qualitative Think-aloud Study of Novice Programmers' Code Writing Strategies. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education*, ITiCSE '14, pages 279–284, New York, NY, USA, 2014. ACM.

[132] David Wood, Jerome S. Bruner, and Gail Ross. The Role of Tutoring in Problem Solving. *Journal of Child Psychology and Psychiatry*, 17(2):89–100, April 1976.

[133] Sylvia da Rosa Zipitra. Piaget and Computational Thinking. In *Proceedings of the 7th Computer Science Education Research Conference*, CSERC '18, pages 44–50, New York, NY, USA, 2018. ACM.