

THE EVOLUTION OF AN ONLINE ENVIRONMENT TO SUPPORT THE STUDIO
BASED PEDAGOGICAL APPROACH FOR COMPUTING EDUCATION

By

ANUKRATI AGRAWAL

A thesis submitted in partial fulfillment of
the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

WASHINGTON STATE UNIVERSITY
School of Electrical Engineering and Computer Science

AUGUST 2009

To the Faculty of Washington State University:

The members of the Committee appointed to examine the thesis of ANUKRATI AGRAWAL find it satisfactory and recommend that it be accepted.

Christopher Hundhausen, Ph.D., Chair

Carl Hauser, Ph.D.

Michael Trevisan, Ph.D.

ACKNOWLEDGEMENTS

I am grateful to my adviser, Dr. Christopher Hundhausen of the Department of Electrical Engineering and Computer Science at Washington State University for his constant support and guidance throughout this project. I also thank him for helping in carrying out the statistical analyses presented in Chapter 6. I am thankful to my committee members, Dr. Trevisan of the Department of Education and Dr. Hauser of the Department of Electrical Engineering and Computer Science, for their support and encouragement. Kyle Ryan and Nick Lewis of the Department of Electrical Engineering and Computer Science helped in implementing online tool (OSBLE) presented in Chapter 3. Jonathan Brown of the Department of Electrical Engineering and Computer Science and Dana Fairbrother of the Department of Education helped in gathering data. Mr. Andy O’Fallon, instructor of CptS 121 course at WSU, to allow me to implement and study my online tool in his course. I would like to thank all the people associated with the CPATH research project, for directly or indirectly helping me to improve my knowledge about studio-based pedagogical approach. This project is funded by National Science Foundation CPATH Award (CNS-0721927). Last, but not the least, I would like to thank my friend, Pawan Agarwal for his unconditional love, support and patience throughout this process.

THE EVOLUTION OF AN ONLINE ENVIRONMENT TO SUPPORT THE STUDIO
BASED PEDAGOGICAL APPROACH FOR COMPUTING EDUCATION

Abstract

by Anukrati Agrawal, M.S.
Washington State University
August 2009

Chair: Christopher Hundhausen

Learning computer science is no longer simply a matter of learning computer programming. Indeed, modern day computing jobs demand design, communication, and collaborative skills as well. Unfortunately, these skills are difficult to teach through traditional lecture-based approaches which do not provide students with adequate feedback or suitable opportunities for collaboration. To address this pedagogical challenge, I have explored the *studio-based* learning approach supported by an online web-based learning environment, one that supports asynchronous reviews and feedback and enables in- and out-of-studio learning and interaction, thereby promoting both collaboration and learning design skills. I recently used this online system in a CS1 course at Washington State University. Observational data collected highlights some key benefits of this system, including, a) improved organization of studio sessions, b) increased student participation and collaboration, and c) the potential to increase outside lab participation. This thesis presents and evaluates this web-based learning environment, and suggests directions for future research into the design of on-line learning environments to support studio-based learning.

TABLE OF CONTENTS

	Page
ACKNOWLEDGEMENTS.....	iii
ABSTRACT.....	iv
CHAPTERS	
1. INTRODUCTION.....	1
2. RELATED WORK.....	8
3. SCOPE AND DESIGN OF FIRST VERSION OF OSBLE.....	17
4. PEDAGOGICAL CODE REVIEWS	31
5. ENHANCEMENTS IN OSBLE TO SUPPORT PEDAGOGICAL CODE REVIEWS.....	37
6. IMPLEMENTATION OF OSBLE IN CS1 COURSE AND MAIN FINDINGS	44
7. FUTURE WORK.....	59
BIBLIOGRAPHY.....	63
APPENDIX.....	69

LIST OF TABLES

1. Code Inspection Issue Categories	34
2. Vignette from a video recording, showing how team logged issue collaboratively	48
3. Categorization of inspection log issues by session (Sp 08 – paper-based).....	50
4. Categorization of inspection log issues by session (Sp 09 – full OSBLE).....	50
5. Mean number of issues logged per group in each code review by treatment	52
6. Statistical comparisons of the two treatments with respect to issues identified in the code reviews.....	53

LIST OF FIGURES

1. Snapshot of steps involved in each cycle of Action Research.....	3
2. Embedded Comment in a web document using CritLink.....	11
3. Snapshot of Kukakuka System, showing Linked ACD.....	12
4. Screenshot of Peer Grader, with links to submissions to be reviewed.....	15
5. OSBLE (Instructor Interface) to create a new course.	23
6. OSBLE (Instructor Interface) to manage students and assignments for a course	24
7. OSBLE (Instructor Interface) to add students to a course	25
8. OSBLE (Instructor Interface) to add students to a course using a class roster...	25
9. OSBLE (Instructor Interface) to remove students from a course.	25
10. OSBLE (Instructor Interface) to create a new assignment	26
11. OSBLE (Instructor Interface) to add a course announcement	27
12. OSBLE (Student Interface) to view assignment and announcement for a course	27
13. OSBLE (Student Interface) to access assignment solutions posted by other students for review	28
14. OSBLE (Student Interface) to review solutions and view comments.	29
15. OSBLE (Student Interface) to add a comment.	29
16. Enhanced OSBLE interface after a student logs in.....	39
17. Enhanced OSBLE interface for a student to view assignments.....	40
18. Enhanced OSBLE interface to access assignment solution posted by other students for review	40
19. Enhanced OSBLE interface to access all files uploaded by a student.....	41
20. Enhanced OSBLE interface to review contents of a file	41

21. Enhanced OSBLE interface to log an issue	42
22. Enhanced OSBLE interface showing highlighted lines of code for which issues have been logged.....	42
23. Pop-up showing the list of all outside-lab issues logged	43
24. Snapshot of the Team Code Review Set-Up with Shared Display	49
25. Snapshot of latest version of OSBLE with profile pictures of students	60

This thesis is dedicated to my aunt, Mrs. Rajeshwari Agarwal, for her unconditional support, guidance and love. Everything, I am today, I owe it completely to her.

I also dedicate this to my cousin, Mrs. Versha Agarwal who was gracious enough to share the love of her mother with me.

CHAPTER ONE

INTRODUCTION

New jobs in the IT field are increasingly emphasizing creativity, design, problem solving and collaborative work (Boyer & Mitgang, 1996). In order to address this need and make computing education more engaging, motivating and community-oriented, at Washington State University, we have been exploring a “studio based” approach for computer science instruction. In this approach, students construct solutions to problems, present those solutions to their peers and instructors, and iteratively improve their solution designs based on peer reviews and feedback from the instructor. We implemented this approach in pre-CS1 and CS1 courses by requiring students to participate in face-to-face studio sessions. Interview and observational data collected in these courses pointed out several practical and logistical problems surrounding face-to-face studio sessions that diminished their educational effectiveness for computer science instruction.

A key question arises: How can we customize a face-to-face studio-based pedagogical approach for computer science instruction?

Context of Research

This thesis is just one piece of a larger multi-institutional research project entitled “Exploring Studio-Based Instructional Models for Computing Education”, and is funded by National Science Foundation CPATH Program (Pathways to Realize Undergraduate Computing Education). Washington State University, Auburn University and University of Hawaii are the three institutions collaborating in this project. The two primary goals of

this bigger project are: a) to refine and evaluate a studio-based pedagogical approach that promotes design-oriented learning and working in collaboration for computing education, and b) to build a community of computing educators interested in applying and further developing the approach.

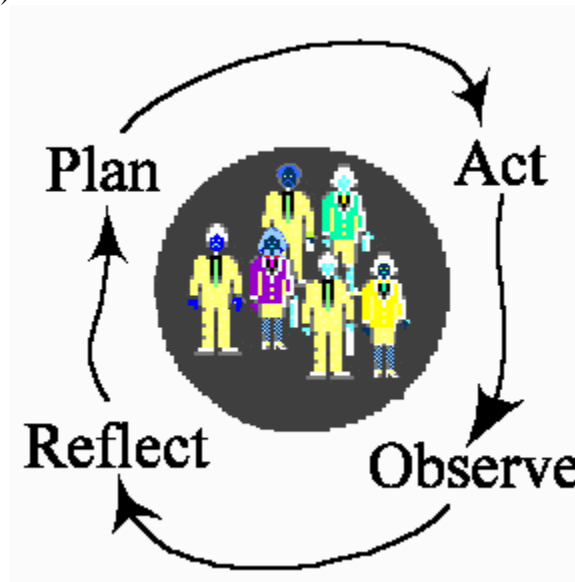
The thesis research presented here encompasses one aspect of this project, which primarily focuses on designing, developing and evaluating an asynchronous online tool to augment face-to-face studio sessions and to tailor studio-based pedagogical approach for computer science instruction. Throughout this thesis, I use the pronoun “we”, to convey the fact that this thesis is part of a larger collaborative effort. While I took a primary role in designing and evaluating the online tool and conducting the empirical studies presented in Chapters 6 of this thesis, this work would not have been possible without the support and collaboration of several others, including graduate students (Jonathan Brown, Dana Fairbrother) and undergraduate students (Nick Lewis, Kyle Ryan) in the VEUPL lab, and my thesis advisor, Dr. Chris Hundhausen.

Action Research

The approach taken by this research project goes beyond traditional research and is more closely aligned with another form of research, known as *Action Research*, a term coined by Kurt Lewin, a social psychologist and educator in about 1944 (O’Brien, 1998). In simple terms, action research can be described as a reflective procedure of continuing research work, undertaken by educators and assisted by professional researchers, which helps them to improve their strategies and practices (Elliott, 1991). Although action research is an alternate research technique that can be applied in any field of research, this term is most commonly used in context of educational research.

Action research uses “a spiral of steps, each of which is composed of a circle of planning, action and fact-finding about the result of the action” (Smith & Doyle, 2007). Each cycle of this spiral consists of the following steps (see Figure 3): 1) Reflect – Think about what we want to focus on, 2) Plan – Plan what to do, 3) Act – Carry out plan, collect evidence, 4) Observe – Observe, monitor and record, and 5) Reflect – Reflect on what has happened to improve further.

Figure 1. Snapshot of steps involved in each cycle of Action Research (see (“action research,” n.d.))



This research falls under the category of action research, as it focuses on improving computer science education practices by experimenting with the studio-based pedagogical approach for undergraduate computer science instruction. The research follows the spiral model of action research, with each semester being one full cycle of this spiral: Reflection, Planning, Action and Observation. Hence, in this thesis, I discuss the progress of research work on a semester-by-semester basis, as in each semester we completed one cycle of following some new process, making observations and then reflecting upon them to decide the course of action for the following semester. As this

thesis proceeds, I gradually discuss the main findings and progress of work for each semester, which is evenly distributed among different chapters.

Studio-based pedagogical approach for computer science instruction

Studio-based learning is an approach to teaching which has been successfully used to teach design skills in architecture and arts education for over 100 years (Lynch, Carbone, Arnott, & Jamieson, 2002). Over the past several years, the studio-based approach has proven to foster creativity and innovation in creative disciplines. In this approach, students come together within so-called “design crits” to present the design of solutions to assigned problems that lend themselves to multiple solution strategies (C. D Hundhausen, Narayanan, & Crosby, 2008). Based on the feedback and discussion generated within these “design crits,” students iteratively improve their solutions. Past experimental studies indicate that discussions mediated by learner-constructed representations are not only educationally beneficial, but also lead to demonstration of critical thinking and stronger sense of community (Hübscher-Younger & Narayanan, 2003). In short, we can say that design studios are a highly interactive, integrative, and iterative experience which acts as a possible catalyst for fostering creativity and complex problem-solving (Birnholtz, Finholt, Horn, & Bae, 2005).

Yet, computer science instruction has traditionally been conducted in a lecture-based format established when computer science departments emerged from mathematics years ago (C. D Hundhausen et al., 2008). The universities involved in the larger project of which this thesis research is a part are now trying to expand the structural boundaries of teaching computer science at the freshman level by implementing studio-based approaches in introductory level courses (C. D Hundhausen et al., 2008).

Over the past several years, Washington State University (WSU) has been one such university to explore the potential for this approach in undergraduate computer science education. In the fall of 2007, we implemented this approach in a pre-CS1 course, which enrolled 90 students. We implemented the studio-based approach by requiring all students to present, for feedback and discussion, their solutions to five progressively more difficult programming projects. Students gave their presentations in five of the regularly scheduled 2 hr. 50 min. lab periods. Due to time constraints, half of the 15 to 20 students in each lab section were required to give presentations on a given day, while the other half were required to write and post peer reviews of selected students' solutions using the Blackboard system ("Blackboard Home," n.d.). To obtain feedback on these sessions, we held debriefing sessions at the end of three of the sessions. In addition, we conducted semi-structured interviews with five students in order to obtain more comprehensive and formal feedback on the approach.

Pitfalls of face-to-face studio-based sessions for computer science instruction

Exit interviews with students conducted at the end of the semester and observational data collected in this course pointed out several practical and logistical problems surrounding these face-to-face studio-based sessions (Anukrati Agrawal & Christopher D. Hundhausen, 2008). Some of the most common problems identified were the following:

(a) *Studio-sessions were not well organized.* Approximately 30 percent of the studio time, according to our estimate, was spent organizing presentations and switching laptops to the projector between these presentations.

(b) *The need for a central repository.* Students wanted access to each other's work outside of lab, and students needed easy access to their own work within lab. A central repository (file server) would make this possible.

(c) *Real time feedback was difficult.* Students said that they found it difficult to provide constructive feedback on their peer's work within the real-time context of a studio session. They felt that they would be able to make more substantial and in depth comments and suggestions if they were afforded the opportunity to review their peers' work on their own time outside of class.

(d) *Written reviews preferred.* Student presenters expressed a preference to receive the reviews of their peers in written form instead of through verbal communication, as this would help them keep track of the feedback and allow them to take it into consideration in future assignments (Freeman & Capper, 1998).

Research Questions

Given this feedback, we wondered whether an online system might help in improving and adapting the studio based approach for computer science instruction. This led to the following research questions:

RQ1: How can we take advantage of an asynchronous online learning system to improve and tailor the studio based approach for teaching computer science?

This question, in turn, led to a second, related research question:

RQ2: What might the design of such a system look like?

In this thesis, we address both of these questions by (a) discussing the ways in which an online learning system can provide features to complement the traditional face-to-face studio based approach, and also tailor it for computer science and (b) presenting the evolution of the design of a specific online learning system to support various studio-based activities for computer science instruction.

Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 discusses the relevant field of research and reviews related work. Chapter 3 presents the initial design and scope of an asynchronous web-based project review system (the Online Studio-Based Learning Environment—OSBLE) to support studio-based learning in computing education. Chapter 4 introduces “pedagogical code reviews” as a way to implement the studio-based approach in computer science courses. Chapter 5 presents the enhancements in the design and scope of OSBLE to support pedagogical code reviews. In Chapter 6, I discuss how we used the OSBLE system to support pedagogical code-reviews in the CS1 course at WSU in spring of 2009, and present the main findings of our evaluation study. Finally, in Chapter 7, I summarize the contributions of this research, and outline directions for future research.

CHAPTER TWO

RELATED WORK

The research presented here develops and explores an on-line system to support the studio-based instructional approach in computing education. In this chapter, I situate the research within the context of three lines of related work: (a) asynchronous discussion, (b) artifact-centered discourse, and (c) on-line systems to support (a) and (b). In this chapter, I highlight the importance of supporting asynchronous discussions and their potential for improving the quality of online feedback and discussion. This chapter also discusses the relevance of *artifact centered discourse* for studio-based computer science instruction and sheds light on all three ways of connecting a referenced document to the related discussion in artifact centered discourse. This chapter also presents a review of some related online tools which support either or both of these features.

Asynchronous Discussion

Research conducted by Davidson-Shivers (Davidson-Shivers, Muilenburg, & Tanner, 2002) examined how graduate students participated in both synchronous and asynchronous discussion. They found that, although students enjoyed both forms of online discussion, asynchronous discussion provided an opportunity for them to give reflective, thoughtful feedback and suggestions and to provide insightful responses to others' opinions and ideas. Asynchronous discussion allows students to read and respond "any time," which gives students more time to reflect and compose a response. This can help in making the learning experience more beneficial and informative, as it by provides students with additional time to reflect, so that they can give more thoughtful responses.

Bhattacharaya (Bhattacharya, 1999) also found that learners preferred asynchronous discussion because it gave them an opportunity to read, craft responses and be better able to think critically before responding.

Also, documented feedback given during web-based asynchronous discussions affords instructors the unique opportunity to assess what has been written and students the unique opportunity to read what they as well as their peers have written as many times as they wish. It also provides instructors with a mechanism to log student's participation in the entire review process. An existing study that uses content analysis for evaluating asynchronous online discussions indicates that the students who were quiet in class were active in the online discussion and that the contribution of all the students significantly improved as the online discussion progressed (Bali & Ramadan, 2007). This kind of asynchronous web based system provides opportunities to the students who may lack confidence to speak in front of their peers or instructor to express their views.

In addition, asynchronous discussion tools also help people to participate remotely in the review process. This helps students who may not be able to attend any of the lab sessions. Online reviews also simulate the conditions of competition, and give students an opportunity to improve their technological and presentation skills. This is supported by the studies of Freeman and Carper (Freeman & Capper, 1998), which showed that an asynchronous web based system enhanced student learning and helped prepare students for effective technology usage in their prospective workplace.

The results of previous studies also raise some potential issues that exist with web-based asynchronous discussions (Black, 2005). Some students feel that they are challenged by the lack of personal contact and the opportunity to get immediate feedback

as compared to traditional classroom discussion. While using such systems, sometimes students get distracted from their central focal point and start discussing other trivial or irrelevant topics (Romiszowski, 1995). Another major issue is difficulty in closure of a discussion. A final response by the instructor may result in closure of an unbound discussion on any topic (Beaudin, 1999).

Artifact Centered Discourse

One of the important contributors to effective learning activities is the use of representational aids (Suthers & Xu, 2002). Students enrolled in studio-based computer science courses will need to share and discuss the artifacts of software engineering (e.g., interface mockups, class diagrams, source code etc.) as they collaborate with faculty and with other students on the design, implementation, and evaluation of their work. Interaction with the artifacts is an essential part of the studio-based learning activity. This should also be true of online system that supports studio-based learning.

Yet online environments usually provide only limited support for *artifact centered discourse* (ACD), a term coined by Suthers and colleagues (Suthers, 2001). Many commercial products for online learning are not designed to support the feature to carry out a discussion in the context of an artifact and the ability to annotate these artifacts. Some tools support annotation of textual artifacts by adding localized comments, but do not support an extended discussion.

Online environments supporting ACD should allow the inclusion of visual artifacts of problem solutions under construction. Participants should be able to browse through the artifacts, and refer to individual artifacts (ideally, parts of artifacts) in their

contributions (Suthers, 2001) (Takeda & Suthers, 2002). Each discussion space should be focused on a single artifact or collection of artifacts, addressing some of the coherence and divergence problems of threaded discussions (Takeda & Suthers, 2002).

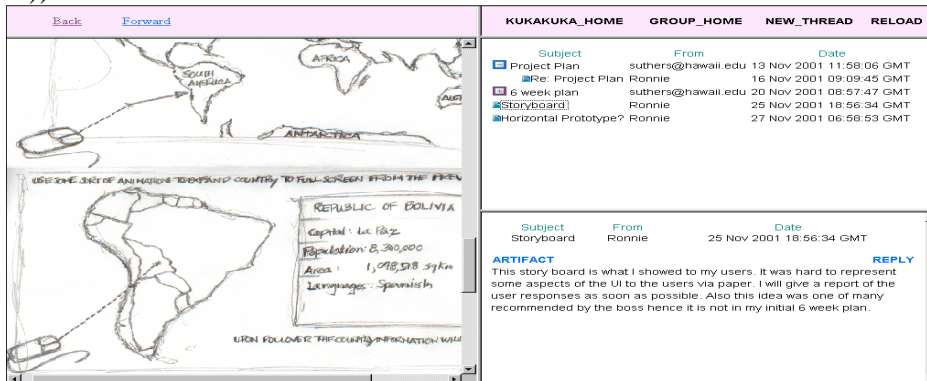
The way in which a referenced document is connected to the related discussions is another important aspect of ACD (Suthers & Xu, 2002). Three primary techniques for ACD are – parallel, linked and embedded ACD. In most of the online learning tools (such as Blackboard and WebCT), the discussion tools and shared artifacts are displayed on entirely different screens. This kind of representation is not conducive to online artifact centered discourse. One way to work around this problem would be to manually open two windows next to each other and place the discussion tools in one window and the artifacts under discussion in the other window. This arrangement is known as *parallel artifact-centered discourse* or *parallel ACD* (Suthers & Xu, 2002). There is no communication or coordination between the discourse and the artifacts, and they are simply displayed in parallel. In an *embedded* discourse representation, comments are directly embedded into the artifact under discussion. Examples of system using embedded discourse representation include the Annotation Engine (Seltzer, 2000), CoNote (Davis & Huttenlocher, 1995) and CritLink (Yee, 1998). Figure 2 presents a screenshot with an annotation embedded to the ACM webpage using CritLink.

Figure 2: Embedded Comment in a web document using CritLink (see (Yee, 1998))



Another approach of connecting a referenced document to the related discussion is known as linked ACD. This approach can be best described as an amalgamation of parallel ACD and embedded ACD. The linked ACD provides a middle way as in this connection approach, the referenced artifact and related discussion are placed side-by-side, and at the same time the chronology of the discussion is also maintained. Examples of linked ACD systems include the Pink (Takeda & Suthers, 2002) and Kukakuka (Suthers & Xu, 2002). Figure 3 below, presents a screenshot, of the Kukakuka system, as an example of Linked ACD. As seen in Figure 3, the left panel displays the referenced document, whereas the right panel displays the related discussion, in a chronological manner.

Figure 3: Snapshot of Kukakuka System, showing Linked ACD (see (Suthers & Xu, 2002))



Related Tools

The system that we propose here is intended to complement the traditional studio based approach and also help in customizing it for computer science instruction. A large body of work shares our interest of supporting asynchronous feedback and discussions (Boyer & Mitgang, 1996; Hübscher-Younger & Narayanan, 2003; “Blackboard Home,” n.d.; Takeda & Suthers, 2002; Suthers & Xu, 2002); however, none (to the best of our knowledge) specifically supports the studio based approach for teaching computer science. For example, WebCT (Burgess, 2003) (now called Blackboard) is an online learning system that is used in many universities for e-learning and provides tools like discussion boards, mail systems and live chat. WebCT differs from our system in that it does not support one of the key aspects of studio-based learning approach—document centered discussion, a central theme for our system.

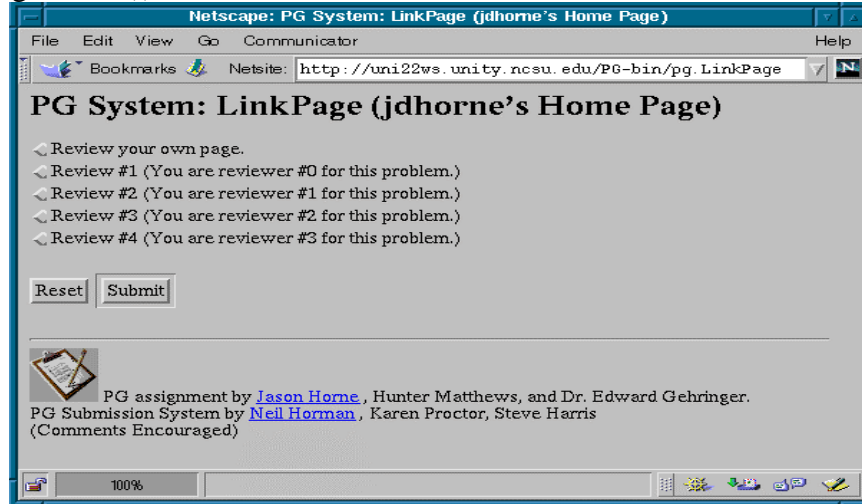
Another application closely related to our proposed system is CAROUSEL (Collaborative Algorithm Representations Of Undergraduates for Self-Enhanced Learning) developed at Auburn University (Hübscher-Younger & Narayanan, 2003). CAROUSEL supports a similar pedagogical approach to teaching introductory computer science courses on algorithms. Using this software, students can create, share, discuss and evaluate multiple graphical representations of their algorithms. The evaluation studies of CAROUSEL and other tools, which showed a significant positive relationship between the constructive and collaborative activities and learning (Lynch et al., 2002; Hübscher-Younger & Narayanan, 2003; (Anukrati Agrawal & Christopher D. Hundhausen, 2008), provides direction and motivation for our proposed system. Our system will take studio-based learning a step further than CAROUSEL, in that it will not be limited to algorithms

courses; rather, it will facilitate the studio based approach for any computer science course that involves designing and programming solutions.

Pink (Takeda & Suthers, 2002) developed at University of Hawaii, supports linking threaded discussion with textual documents. Using Pink, one can embed anchors, in form of icons, in the textual document. These anchors can later be used to refer parts of the document in the discussion notes. Kukakuka (Suthers & Xu, 2002) developed at University of Hawaii, is another system that supports artifact centered discourse. It does so by coupling discussion groups with online artifacts in the form of web pages. Kukakuka is limited in its scope as it supports discourse around web pages. Although the current implementation of our online tool only supports artifact centered discourse around textual documents containing student's code solutions, our system is designed in a way that it will overcome this limitation, and the future versions of our system will be able to support discourse centered on any kind of artifact.

Peer Grader (PG) (Gehring, 2001) is another web-based system to facilitate asynchronous on-line peer reviews, developed at North Carolina State University. PG allows students to submit their assignment solutions to the system. Once the assignments are submitted, PG allows other students to review and grade the assignments of their peers. One of the unique features of PG is that it allows authors to revise their work based on the peer reviews, and to resubmit their work again for review. This cycle of re-submissions, helps authors and the reviewers to communicate with each other. Figure 4 below, presents a screenshot of Peer Grader tool, showing a page with links student assignment submissions to be reviewed.

Figure 4: Screenshot of Peer Grader, with links to submissions to be reviewed (see (Gehring, 2001))



Because we that allowing students to resubmit their work motivates students to reflect and act upon the reviews given to them, the system presented in thesis also facilitates student re-submissions of their work.

Another tool developed by computing educators to facilitate asynchronous on-line peer reviews is Read, Review and Access System (RRAS). RRAS allows students to submit their assignment online and also allows other students to review and assess solutions posted by other students. RRAS differs from the other tools discussed above in that it allows students to conduct reviews based on a rubric provided by the instructor. RRAS differs from our online tool in that the reviews provided by RRAS are anonymous, whereas in the tool presented here the reviewers are known. We identify the reviewers in our system to make them accountable for their work and also to increase ownership.

To summarize, while most of the research work and online learning systems discussed above support some of the features required for supporting studio-based learning in computing education (like asynchronous discussions, artifact centered discourse and online review), the research presented here differs from the above in that it

attempts to design an online tool that is specifically tailored to support studio-based learning in computing education and will support all the aspects of studio-based learning approach. In next chapter, Chapter 3, I present the preliminary design and scope of the first version of this online tool, called OSBLE.

CHAPTER THREE

SCOPE AND DESIGN OF FIRST VERSION OF OSBLE

As discussed in Chapter 1, the data collected for assessing the effectiveness of the studio-based approach in CS1 course at WSU in Fall 07 semester pointed out several practical and logistical problems surrounding these face-to-face studio-based sessions. Based on this feedback, we decided to build an online tool in order to explore whether a tool might help in solving any of those issues. In this chapter, I present the initial design and scope of an online tool, OSBLE (Online Studio-based Learning Environment), to support studio-based sessions.

OSBLE is intended to complement the traditional studio-based approach and also help in customizing it for undergraduate computer science instruction. Thus, the main stakeholders of this system are computer science instructors and undergraduate students enrolled in preliminary CS courses. Future versions of OSBLE may also support more courses spread across various engineering disciplines and beyond. In this chapter, I discuss the preliminary data gathering and initial design and scope of the system. I also discuss the functionalities supported and present the interface developed for the first version of OSBLE.

We have followed user-centered design process for designing OSBLE. ‘User-centered design’ (UCD) is a broad term, that originated in the 1980s in Donald Norman’s research laboratory at the University of California San Diego. As the name suggests, the UCD design principle encourages the participation of end-users in the design process. There are several ways in which users can participate in the UCD process. One way of

involving users in the UCD process is by requiring them to participate actively in the requirements gathering and usability testing processes. Another way is to consult the users during the design process to elicit their feedback. We designed OSBLE by involving users in the requirements gathering process, and also by having them participate in usability studies. In the following subsections, I describe how we designed the first version of OSBLE employing UCD.

Preliminary Data Gathering for establishing the initial scope of OSBLE

Performing data gathering for this system was one of the most challenging tasks, as this was a novel software idea and not everyone easily understood the scope of our proposed system. It was important that we first explained the concept of studio-based approach and high-level scope of the system to stakeholders and then elicit their feedback and suggestions for establishing the requirements of the system.

Two main user groups for OSBLE are the instructor and students. We made sure to include representatives from both these user groups as gathered data. Based on this analysis and to add credibility to our findings, we decided to use the *data triangulation technique* (Thurmond, n.d.) for data gathering: (a) first perform face-to-face interviews with CS instructors, (b) conduct focus groups with undergraduate students enrolled in preliminary CS courses, and finally (c) conduct a low-fidelity paper prototyping of the system. These prototypes were based on the data gathered from (a) and (b). In low-fidelity prototyping studies, both classes of users (instructors and students) performed the core tasks using our low-fidelity prototypes. This step helped in covering any requirements missed in the earlier phases of data gathering.

What follows is a detailed description of how we performed each data gathering activity and its key results.

Interviews

The interviews were conducted to gather requirements data from instructors. We followed a semi-structured format for conducting interviews, as this gave space for open-ended questions arising from the users' responses. We interviewed two Computer Science TAs. In addition to these TAs, we also interviewed a high school science teacher. Why interview an instructor outside our initial target audience? First, computer science instructors are typically much more comfortable figuring out software than other instructors, so a better test of usability can often come from someone less familiar with software technology. Second, in the long term, we want this system to be applicable for other subjects beside computer science. We intend to make the system so flexible that instructors in a wide variety of subjects will want to try this studio-based approach to learning.

The interviews were one-on-one, face-to-face and semi-structured in nature and lasted about 25 minutes on an average. All the interviews were conducted in an "office-like" setting. The interviewees were first given a brief description of the studio-based approach and how it is being conducted for certain undergraduate courses at WSU. Then we introduced the interviewee to the concept of our proposed system and its high level scope, after which they were asked a set of open-ended questions for establishing the requirements of the system. Some of the key results of these interviews included: (a) instructors should have privileges for adding or removing students from a class, (b) students should be able to submit multiple files for an assignment; (c) students should be

able to upload files of various formats (.doc, .cpp, .ppt etc), (d) instructors should be able to maintain an evaluation rubric using his system, and (e) all the project files submitted by the students should be *locked* (that is, students would not be able to update them, past the assignment date) once the due date is passed.

Focus Group

The focus group was conducted to gather data from the students. We selected a group of four CS students, as our system is primarily targeting facilitating the studio-based approach for computer science instruction. We also included one student from the electrical engineering department, to get a wider perspective (this will be helpful if we plan to extend our system to support other engineering disciplines as well). The focus group lasted for about 30-40 minutes.

The focus group was conducted in a quiet, “classroom-like” setting. The interviewees were first given a brief description about the studio-based approach and how it is being conducted for certain undergraduate courses at WSU. For this introduction, we made use of blackboard and chalk, and described the setting of a studio-based session and an idea about our proposed system using rough drawings. Using these drawings, we introduced the participants to the concept of our proposed system and its high level scope, after which they were asked a set of open-ended questions regarding their expectations from such a system. Some of the key results of this focus group included: (a) students should have access to projects of other students before the classroom studio-session, so that they can go prepared for the session, (b) students should have access to projects of other students after the studio-session is over, (c) students should have the flexibility to update their assignment files up until the deadline is passed, (d) students

should be able to submit multiple files for an assignment, (e) students should be able to upload files of various formats (.doc, .cpp, .ppt etc.)

Low Fidelity Prototyping

Low fidelity prototyping was performed after conducting the interview and focus groups. We first created rough low fidelity paper-prototypes (using pen and paper) based on the data gathered from the interview and the focus group. Then we conducted a study in which users (both instructors and students) performed core tasks using the low-fidelity paper prototypes. This step helped us to cover any requirements missed in the previous two phases of data gathering. We conducted low-fidelity prototyping using the same candidates who had participated in our interviews and focus groups. By doing so, we were sure to advance our data gathering process by avoiding redundant suggestions. This also helped to conduct the entire process in a more efficient and effective manner as the audience was already aware of the system.

This prototyping was conducted in a quiet, “office-like” setting. We first gave the user a description of the task that he had to perform and then asked them to perform the task using paper prototypes. While performing this low-fidelity prototyping, we asked questions such as: (a) What you would expect the system to do next? (b) What kind of response do you expect after clicking here? (c) Did you easily interpret the system response? Some of the key results included: (a) The projects should be grouped by project number, and (b) instead of directly opening the code file of a student, the system should provide links to all the code sections in the code.

Conceptual Design Model

This system is designed around two similar conceptual models: the model giving instructions and the model of exploration and browsing (Preece, Rogers, & Sharp, 2001). At the core, the system is run by giving instructions through menu items, buttons, and other controls. Throughout the website, the system provides a series of menus. Each menu item is an instruction: “Create a New Assignment.” “Manage Students and Classes.” “View Past Assignments.” Links and buttons on screen also are instructions: “Go to Class Roster.” “Add a Comment.” “Upload File.” Thus the system is built around the idea that the user will specifically instruct the system where to go and what to do through predefined controls. Clearly, we have designed this all in a WIMP interface (Windows, Icons, Menu, and Pointing Devices).

When entering the studio-based peer review process, the model becomes more complicated. Although the model of giving instructions does not go away, a different model takes center stage: exploring and browsing. For each completed assignment, the user will be able to view all the files of each student in the class, and the user may browse by student, by file, or by section. Here students learn by exploring: scanning through various sections, browsing existing comments, and interacting where appropriate.

Altogether, our conceptual model is based on the activities of interacting with the system and with other students through the system. We avoid interface metaphors in our conceptual model. The interface model could have been designed around an image of a classroom with studio tables. However, since the online studio is significantly different from the in-class studio approach—namely, the online approach is entirely asynchronous—we believe that it would only confuse the user to model the interface in such a manner. We also avoid the model of direct manipulation. Unlike image editing

and word processing, which are clearly manipulative tasks, the task of peer reviewing is abstract and community-based. In the end, if end users form a conceptual model similar to the activity-based model just described, our interface will have succeeded.

Functionalities supported by first version of OSBLE

What follows is a list of all the features supported by first version of OSBLE and a snapshot for their respective user interfaces:

- *An instructor is able to create new course, including relevant details, course syllabus and course schedule (Figure 5). To create a new course, the instructor is able to specify the following: (a) the prefix of course; (b) the number of course; (c) the name of course; (d) the year in which it is offered; (e) the semester in which it is offered. The system allows the instructor to provide the syllabus and schedule of the course either through a hyperlink to a webpage or by uploading a document.*

Figure 5. OSBLE (Instructor interface) to create a new course

Studio Based Learning Welcome, Anu Agrawal (Instructor)
Wednesday January 21, 2009 My Courses Help Logout

Create New Course

Fields marked with a * are required

Course Prefix *

Course Number *

Course Name *

Course Semester *

Course Year *

Course has multiple sections

Number of sections

Students can view other sections

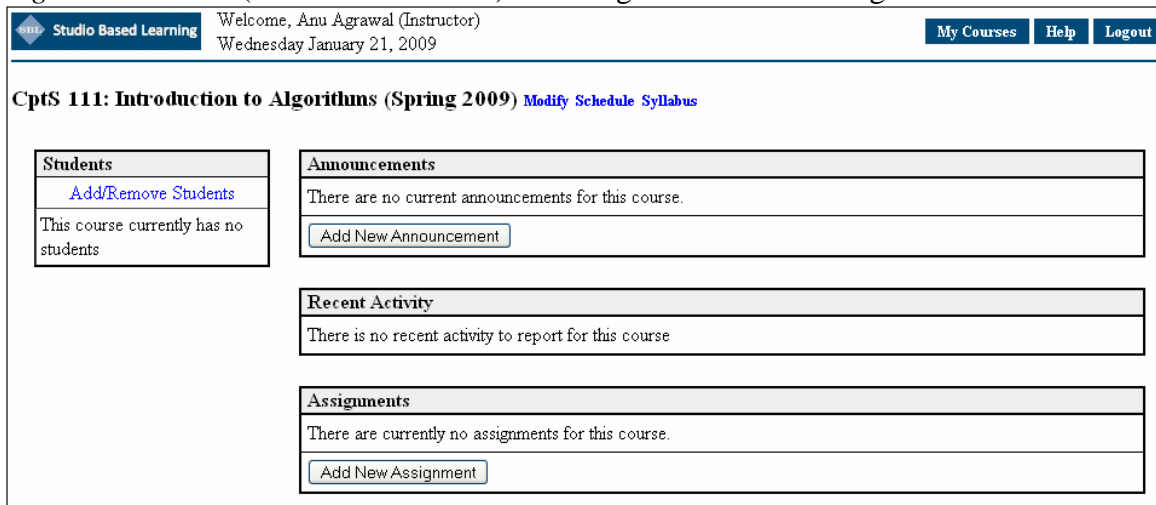
Course Syllabus Url Upload

Course Schedule Url Upload

- Once a new course is created, an instructor is able to add/drop students to/from that course, add assignments and specify announcements for that course (Figure 6).


From the course detail page, the instructor is able to get access to the following: (a) link to a page, from where he can add or drop students to that course; (b) link to a page that allows adding announcements for that course; (c) link to a page that allows adding a new assignment for that course.

Figure 6. OSBLE (Instructor interface) to manage students and assignments for a course




- An instructor is able to add and remove students in the system. An instructor is able to add students to the system (Figure 7), and later to remove any students he or she previously added. When adding a student, the instructor must specify first name, last name, student id number and student's email address. (The student's initial password will be the student id number, but the student must change it upon first login.)

Figure 7. OSBLE (Instructor interface) to add students to a course

 Studio Based Learning		Welcome, Anu Agrawal (Instructor) Wednesday January 21, 2009		My Courses	Help	Logout	
Add a Student Add Multiple Students Remove Students	<h3>Add Student to CptS 111</h3>						
	First Name <input type="text" value="Nick"/>	Last Name <input type="text" value="Lewis"/>		Student Number <input type="text" value="12121212"/>	Email Address <input type="text" value="nlewis@wsu.edu"/>		
	<input type="button" value="Search"/>	<input type="button" value="Create New Student"/>					


The instructor has the option to either add one student at a time or add multiple students to a course by uploading a class roster file, containing the details of multiple students (Figure 8).

Figure 8. OSBLE (Instructor interface) to add students to a course using a class roster

 Studio Based Learning		Welcome, Anu Agrawal (Instructor) Wednesday January 21, 2009		My Courses	Help	Logout
Add a Student Add Multiple Students Remove Students	<h3>Add Students to CptS 111</h3>					
	Student List <input type="text"/>	<input type="button" value="Browse..."/>				
	<input type="button" value="Submit"/>					

The instructor can also remove one or multiple students from a course using the Remove student option (Figure 9).

Figure 9. OSBLE (Instructor interface) to remove students from a course

 Studio Based Learning		Welcome, Anu Agrawal (Instructor) Wednesday January 21, 2009		My Courses	Help	Logout
Add a Student Add Multiple Students Remove Students	<h3>Remove Students from CptS 111</h3>					
	<div style="border: 1px solid gray; padding: 2px;"> 22221111 - Pawan Agarwal ▲ 12121212 - Nick Lewis ▲ 11112222 - Kyle Ryan ▼ </div>					
	<input type="button" value="Remove Students"/>					

- An instructor is able to specify and edit a new assignment, including relevant details, required files, and required function in each file (Figure 10). To create a new assignment, the instructor is able to specify the following: (a) the name of the project;

(b) an evaluation rubric; (c) a description of the project; (d) the start date; (e) the due date; (f) the files required; (g) optionally, in each text file, the set of function names to be embedded in code. At any time after creating the assignment, the instructor is able to go back and edit any of the above fields.

Figure 10. OSBLE (Instructor interface) to create a new assignment

The screenshot displays the OSBLE instructor interface for creating a new assignment. The main form is titled "Create Assignment for CptS 111" and includes the following fields:

- Assignment Title ***: Battle Ship Project
- Start Date**: 01/21/2009 at 15:05 am
- Due Date ***: 01/30/2009
- Assignment Description ***: C:\Anu\O
- Evaluation Rubric**: C:\Anu\W
- Other Description Files**: (empty)
- Required Assignment Files**: << No files specified >>

A modal dialog box is open for adding a file. It contains the following information:

- Name of File**: placeShips.cpp
- Type of File**: Code
- Required functions within file**:
 - Function: (empty)
 - Buttons: Add >>, << Remove
 - Function list: AskLocation, CheckAvailability
- Buttons**: OK, Cancel

At the bottom of the main form, there are buttons for "Add File...", "Modify File", "Remove File", "Create Assignment", and "Cancel".

- An instructor is able to specify and edit a new announcement for a course (Figure 11). To create a new announcement for a course; the instructor is able to specify the following: (a) the announcement title; (b) the announcement text. At any time after creating an announcement, the instructor is able to go back and edit or delete the announcement.

Figure 11. OSBLE (Instructor interface) to add a course announcement

The screenshot shows the 'Create New Announcement for CptS 111' form. At the top, there is a navigation bar with 'Studio Based Learning' on the left, 'Welcome, Anu Agrawal (Instructor)' and 'Wednesday January 21, 2009' in the center, and 'My Courses', 'Help', and 'Logout' on the right. The main content area has the title 'Create New Announcement for CptS 111'. Below the title, there are two input fields: 'Announcement Title' with the text 'Welcome to CptS-111' and 'Announcement Text' with the text 'Hello Students. You all are welcome to the CptS-111 course'. At the bottom of the form, there are three buttons: 'Submit', 'Reset', and 'Cancel'.

- A student is able to read announcements and assignment for the courses he has registered (Figure 12). When an instructor posts an announcement for a course, a student is able to view that course announcement. Also, when an instructor creates a new assignment, a student is able to view all of the information entered by the instructor.

Figure 12. OSBLE (Student interface) to view assignment and announcement for a course

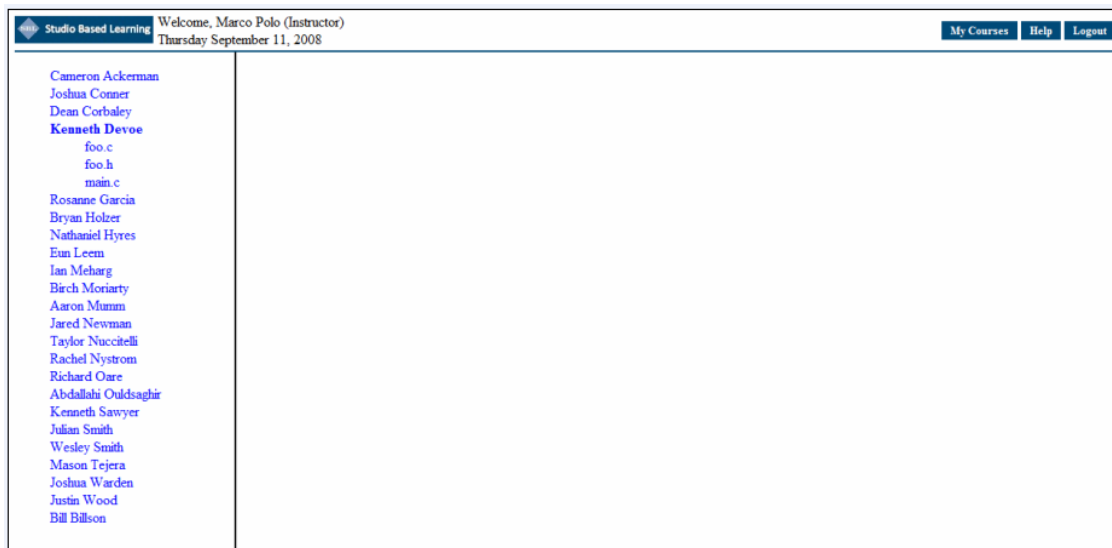
The screenshot shows the student interface for 'CptS 111 : Introduction to Algorithmic Problem Solving (Fall 2008)'. The top navigation bar includes 'Studio Based Learning', 'Welcome, John Doe (Student)', 'Thursday September 11, 2008', and 'My Courses', 'Help', 'Logout'. The main content area is titled 'CptS 111 : Introduction to Algorithmic Problem Solving (Fall 2008) Schedule Syllabus'. It features three sections: 'Announcements' with a 'Welcome 09/11/2008' message, 'Recent Activity' with the message 'There is no recent activity to report for this course', and 'Assignments' with a link to 'Hello World (Due September 19, 08)[Submit]'.

- A student is able to upload completed assignment files. The student is able to upload the required files for an assignment. When a file is uploaded, the system will scan it to verify that the instructor's required functions, if any, are present. If the system finds

a problem with the annotations, it notifies the user and points out the problem or problems in the text. The student may then correct the errors and try again.

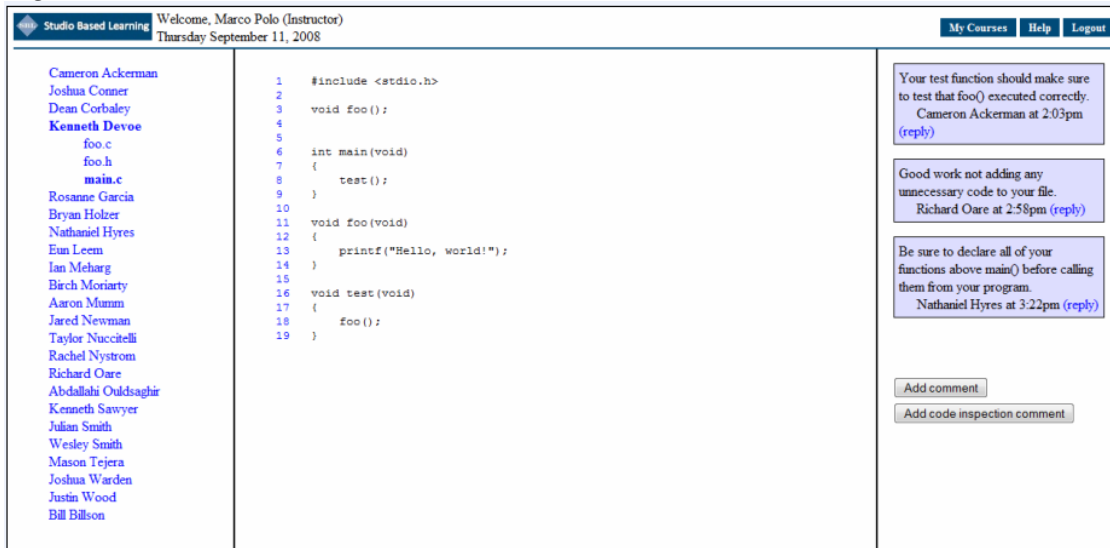
- *A student is able to view other people's assignments.* The student should be able to view the completed assignments over other students who have locked the assignment. After the deadline, this means all students who turned in work. The student can view the uploaded text assignment by another student (choose a student and then a file or function within a file; see Figure 13). A student can also view his or her own work in this manner. All text-files appear in the browser with numbered lines.

Figure 13. OSBLE (Student interface) to access assignment solutions posted by other students for review



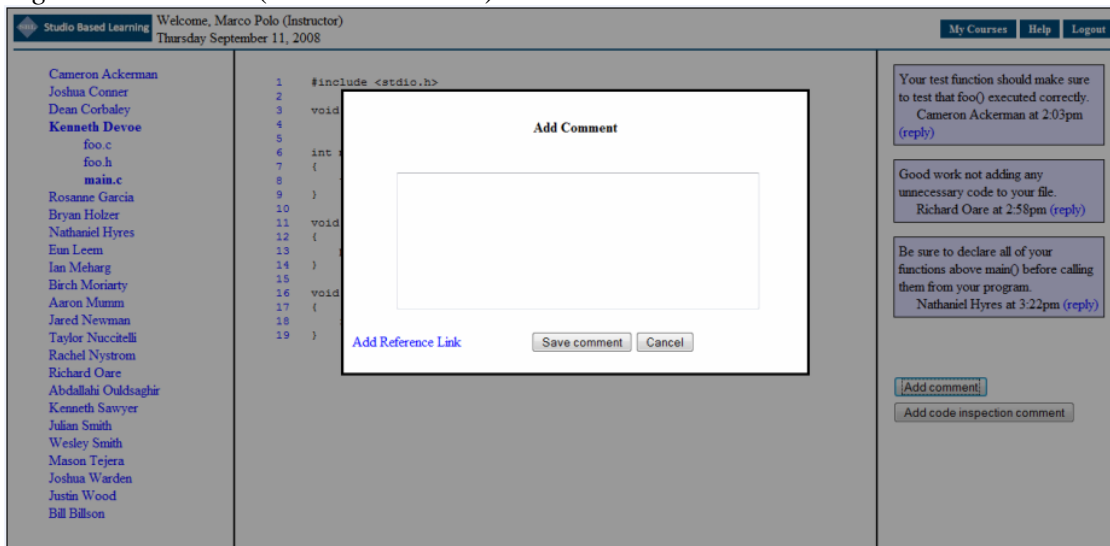
- *A user is able to view and post a comment on a particular section of a text file* (Figure 14). While viewing another student's work, a user is able to leave comments. A user can leave comment to a specific section. These comments will be plain text only. These new comments will all appear next to the relevant function of text. The user may click any comment to view the full comment text. Here they can also reply to an existing comment, generating a mini-thread.

Figure 14. OSBLE (Student interface) to review solutions and view comments



These comments will all appear next to the relevant function of text. The user may click any comment to view the full comment text. Here they can also reply to an existing comment, generating a mini-thread (Figure 15).

Figure 15. OSBLE (Student interface) to add a comment



To summarize, the data collected in our Fall 2007 study in which we implemented the studio-based approach for a pre CS1 course at WSU (discussed in detail in Chapter 1) motivated the need for building an online tool to support studio-based approach and tailor

it for computer science instruction. In this chapter, I discussed the preliminary data gathering techniques employed for establishing the basic requirements of such an online system. This chapter also presented the initial design and scope of this software and described the user interface and functionalities supported by the first version of the software.

While we were working on the design and implementation of first version of OSBLE in Spring 2008, we also studied a new method of implementing studio-based approach (Pedagogical Code Reviews) in the Spring 08 offering of the CS1 course at WSU. . The next chapter describes pedagogical code reviews and how we implemented them in a CS1 course at WSU.

CHAPTER FOUR

PEDAGOGICAL CODE REVIEWS

The two key activities that are emphasized in studio-based instructional approach are: (a) construct solutions to assigned problems and (b) present solutions to instructors and peers for feedback and discussion. In order to adapt this approach for lower-division computing courses that traditionally emphasize individual programming skills, we tried two different implementation approaches at Washington State University: In the first approach, (discussed in detail in Chapter 1), we required all the students to present their solutions to programming assignments to all their classmates and instructor for feedback and discussion. Other students gave verbal feedback at the end of the presentation and also submitted written feedback after the lab was over. In second approach, (discussed in detail in this chapter), we required students to engage in an adapted form of formal code inspections called pedagogical code reviews.

About Pedagogical Code Reviews

Pedagogical code reviews are an adaptation of formal code inspections for use in undergraduate computing courses (C. Hundhausen, A. Agrawal, Fairbrother, & Trevisan, 2009). In pedagogical code reviews, we require a group of three to four students, who are led by a trained moderator (C. Hundhausen et al., 2009), to (a) walk through segments of each other's programming assignments, (b) check the code against a list of best coding practices, and (c) discuss and log issues that arise. The moderator is a senior undergraduate or graduate computer science student who is trained to lead such group code reviews.

Participating in pedagogical code reviews enables students to exercise a variety of skills like (a) computer programming (C. Hundhausen et al., 2009), (b) critical review of their own and others' code - by having them review each other's code against an established checklist of best practices (C. Hundhausen et al., 2009), and (c) communication and teamwork by having them review each other's work as part of a team (C. Hundhausen et al., 2009).

Many educators have exercised different peer review approaches in a variety of different courses in upper-division computer science courses. For example, (Kern, Saraiva, & dos Santos Pacheco, 2003) implemented an "in class conference" in an upper-division database course. This approach was designed after a professional academic conference. Anewalt (Anewalt, 2005) also talks about her implementation of peer reviews in a senior software engineering course.

Computer science educators often agree on the possible pedagogical advantages of peer reviews. Some of the frequently cited benefits of peer reviews include: a) the process of participating in team reviews prepares the students to provide constructive criticism (Anewalt, 2005), b) it also prepares the students to cope with criticism of their own work (Anewalt, 2005), c) it helps to develop soft skills required to work in teams (Trytten, 2005), and d) it promotes deep learning and better understanding of the subject matter (Gehringer, Chinn, Pérez-Quinones, & Ardis, 2005). Peer reviews aim to provide the benefits discussed above, as they provide students with learning and assessment opportunities are not offered by traditional teaching methods. Moreover, pedagogical code reviews strive to build student's communication skills and promote a sense of community skills, by having them review each other's work on a team.

Implementation of pedagogical code reviews at WSU¹

In Spring of 2008, we implemented pedagogical code reviews in one of the five sections of CptS 121 “Program Design and Development,” the CS 1 course at WSU. Twenty-one students participated in structured pedagogical code reviews. These code reviews were conducted for three programming assignments (out of total 7 programming assignments), and were carried out in regularly scheduled two hour and fifty minute lab periods. These reviews were conducted in eighth, eleventh, and thirteenth weeks of the fifteen week semester.

For these code reviews we divided students into teams of three to four students. These teams were formed in a way that they were roughly equivalent in terms of the abilities of their members. Each team of undergraduate students was accompanied by a Moderator, during the code review session. Moderators were graduate students who had been trained to moderate the code reviews. Moderators were responsible for leading the team reviews and for anchoring the team members by following a well-defined procedure.

The pedagogical code reviews implemented in Spring 2008 semester were completely pen-and-paper based and were not supported by any kind of online tool. Students were asked to bring five print-outs of their programming assignment solutions to the three labs in which the code reviews were conducted: one for each team members and one for the moderator.

While participating in the code reviews, the students were required to play the following roles (see, e.g., (Gilb, Graham, & Finzi, 1993)):

¹ This study was originally reported in (C. Hundhausen et al., 2009). It is summarized here.

Reader — the person who reads the Author’s code solution aloud.

Inspector — the person who takes a leading role in inspecting each line of code against the provided code review checklist.

Recorder — documents issues that arise in the inspection in a provided inspection issue log.

All the team members were given the opportunity to play all of the above mentioned roles. Every time the team reviewed the code of a new student, other students were required to switch from the role they played last time to a new role (C. Hundhausen et al., 2009).

Reviews of each programming solution were done against an established checklist of coding best practices (Wiegers, 1995). This checklist included issues classified in seven general categories (see Table 1). In order to check for the assignment-specific requirements, this list was augmented with the requirements for the specific programming assignment (issue category 1 in Table 1). The review of each student’s code solution in the team resulted in a detailed log of issues.

Table 1. Code inspection issue categories

#	Issue Category
1	Code meets assignment requirements
2	Structure and design
3	Documentation, standards, and formatting
4	Variables and constants
5	Arithmetic operations
6	Loops and branches
7	Defensive programming

The lab code reviews were usually scheduled two days after the assignment due date. This provided students with enough time to review their code independently, before

participating in team reviews in lab. The code reviews of each team were conducted, following a detailed protocol enforced by the trained moderators. Code review of every individual in the team was limited to 30 minutes. At the beginning of each review, the Moderator assigned roles (Author, Reader, Inspector and Recorder) to team members. After this, the person assigned the role of the Reader, read the code of author line-by-line. While the reader read the code, anyone who found an issue with the code was free to jump in between and raise that issue. If valid, the team classified the issue according to its severity and type. Then the recorder documented the issue on a sheet of paper, known as the issue log sheet. At the end of each review, the recorder read aloud all the issues to the team members. This was done to make sure that everybody in the team was on the same page and to make sure that all the issues have been recorded properly.

For my thesis work, I was particularly interested in studying whether these completely paper-based code reviews were self-sufficient or if there was a need to augment and support these reviews with some kind of an online tool. So, the main findings for my thesis research came from the observational data collected this semester.

The observational data collected in this course pointed out several practical and logistical problems regarding the implementation of completely paper-based pedagogical code review sessions. Some of the most common problems identified were:

(a) Format of code reviews lacked an appropriate means to allow student to do “ahead of time” reviews, which is one of the keys to the success of traditional code inspections (Gilb et al., 1993),

(b) Review comments and issues logged were not available outside the class to all the students for future reference (Anukrati Agrawal & Christopher D. Hundhausen, 2008),

(c) Requiring five hardcopies of each student's assignment solution f was both cumbersome and eco-unfriendly (C. Hundhausen et al., 2009), and

(d) With three to five students in each group, the code review process seemed to become redundant as similar issues were found (C. Hundhausen et al., 2009).

In order to address problems (a), (b) and (c), we decided to enhance OSBLE to support and augment pedagogical code review sessions. In next chapter, I discuss in detail the enhancements made to OSBLE to support pedagogical code reviews. These enhancements were implemented in OSBLE during the Fall 2008 semester.

Also, as part of the overall CPATH project, data was collected in this semester to evaluate the effectiveness of pedagogical code reviews as a studio-based activity. Issue logs prepared by the teams while participating in these code review sessions and the exit questionnaires completed by all the students at the end of each review session were analyzed. I have used some of these data for my thesis, as a means to compare completely paper based code review sessions of Spring 2008 with code review sessions supported by OSBLE, implemented in Spring 2009. The results of this comparison are discussed at length in Chapter 6, which talks about main findings of Spring 2009.

CHAPTER FIVE
ENHANCEMENTS IN OSBLE TO SUPPORT PEDAGOGICAL CODE
REVIEWS

Realizing the potential value of pedagogical code reviews as a studio-based activity in computer science instruction, we decided to enhance OSBLE to support pedagogical code reviews. The key tasks facilitated by a new version of OSBLE to support pedagogical code reviews include:

- Accessing and logging in to the system (Tasks 1 and 2 in next section)
- Viewing an assignment to be reviewed (Task 3)
- Viewing the code solutions of a student for a given assignment (Task 4)
- Starting an in-lab review (Task 5)
- Logging issues with a team member's assignment solution within the in-lab review (Tasks 6 and 7)
- Finalizing an in-lab review (Task 8)

In order to facilitate pedagogical code reviews, OSBLE supports three user interfaces: (a) student; (b) instructor; and (c) code moderator. Instructors post the assignment description through the system. Students submit code solutions to be reviewed through the system. Once they have done so, they then have the option to “lock” their assignment. Once solutions are "locked" - student may no longer modify them, and they now have access to the code solutions of their peers. This provides opportunity for a period of on-line code review. Students are encouraged to view the solutions of the members of their review team, to identify issues with those solutions, and to log those issues in the system, ahead of time, before participating in face-to-face, in-

lab review sessions. These on-line reviews are meant to prepare students for the face-to-face code review sessions that take place during the regularly scheduled lab periods.

OSBLE facilitates face-to-face reviews as well. When a review of a given team member is about to begin, the team moderator can use OSBLE to specify the role that each team member will play within the review. Team members can use OSBLE to view each other's code, to view the issues that have been logged prior to the face-to-face review, and to log issues identified within the face-to-face review. After the face-to-face reviews are over, OSBLE allows students to view the review logs associated with the review of their solutions as well as the logs associated with the review of other student's solution. Finally, OSBLE allows students to resubmit their assignments based on the results of the face-to-face review. (Please refer to appendix for a detailed tutorial on performing key tasks with student interface in this enhanced version of OSBLE).

Walkthrough

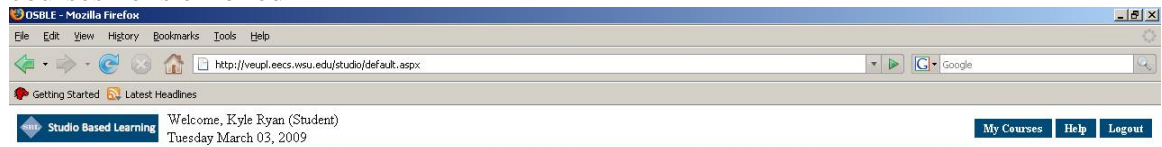
To illustrate how students might use enhanced version of OSBLE, I now step through a sample task scenario for one of the key tasks supported by OSBLE: *Review the assignment solutions posted by one of your team member and log an outside-lab issue for a particular line in his code. Once the outside-lab issue has been logged, transfer this issue logged outside lab into the list of in-lab issues (For second part of this task, assume that your group met for face-to-face code review in lab, recognized the same issue as a group and hence decided to transfer it from outside-lab to in-lab issue list)*

This task can be broadly classified into three parts – 1) Accessing the assignment solution of a particular team member and viewing a particular solution file, 2) Logging an outside-lab issue on a particular line in his this assignment solution file, and 3)

Transferring the issue logged outside-lab into the list of in-lab issues. Below, I present a detailed walkthrough to accomplish all of the above tasks, by demonstrating the subtasks and a snapshot of the system while performing them:

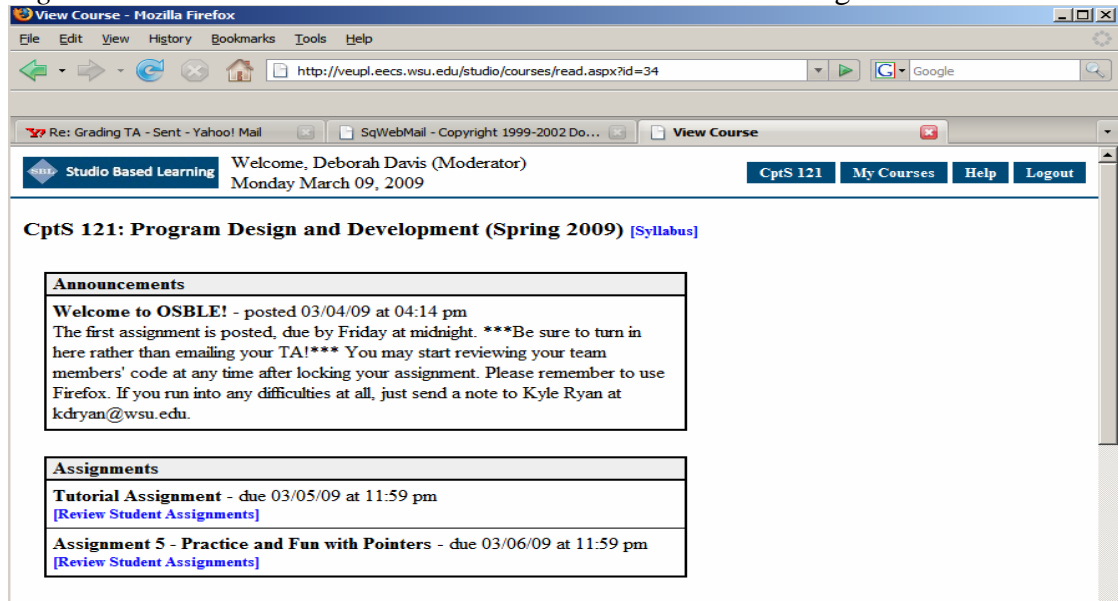
1. *Accessing the assignment solution of a particular team member and viewing a particular solution file.* First, we need to log into the system as a student, using our username and password. This can be done by typing the URL <http://veupl.eecs.wsu.edu/studio/> in the address bar of a browser. After logging in to a student account, we can find a link to all the courses in which we are enrolled (Figure 16). We then select the particular course for which we have to do the review. This can be done by clicking on the course hyperlink.

Figure 16. Enhanced OSBLE interface after a student logs in and views the courses he is enrolled in



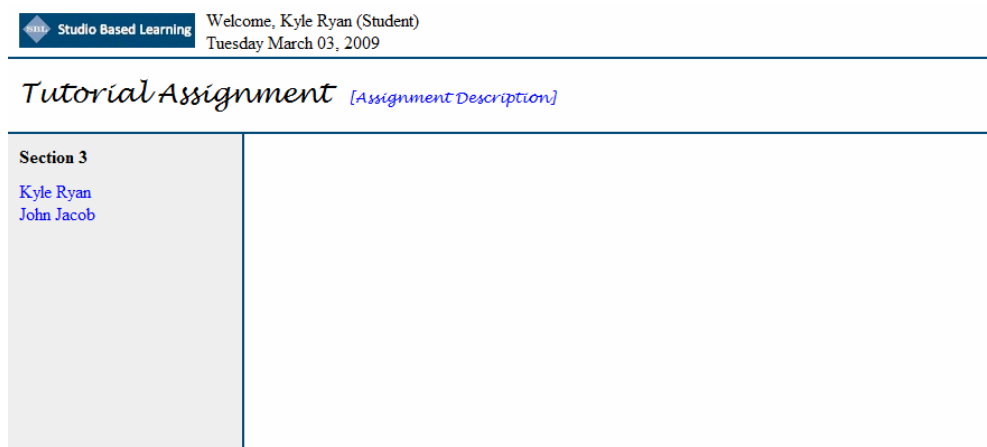
- a. Once we select a particular course by clicking on it, links to all the assignments posted by the instructor for that particular course become accessible. Next, we select the particular assignment for which we have to do the review. This can be easily done by clicking on the assignment hyperlink (Figure 17).

Figure 17. Enhanced OSBLE interface for a student to view assignments



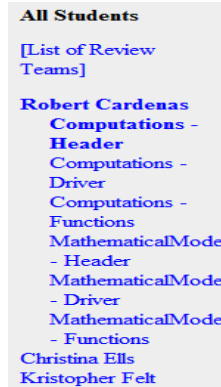
- b. Once we select a particular assignment, links to names of all the students who have uploaded and locked their solutions for that assignment are displayed. Next, we select a particular student whose code we want to review, by clicking on his name (Figure 18).

Figure 18. Enhanced OSBLE interface to access assignment solutions posted by other students for review



- c. After selecting a particular student, links to all the assignment solution files posted by that student for that particular assignment are displayed (Figure 19).

Figure 19. Enhanced OSBLE interface to access all the files uploaded by a student



- d. Next we can select the particular file, which we want to review, by clicking on it. Once, we select a file, its contents become visible in the right panel (Figure 20). With this we accomplish the first basic task, of accessing the file which is to be reviewed.

Figure 20. Enhanced OSBLE interface to review contents of a file

```

1  #include "functions.h"
2
3  int main (void)
4  {
5      int num = 0,
6          primeIndicator = 1,
7          multipleIndicator7 = 0,
8          multipleIndicator11 = 0,
9          multipleIndicator13 = 0,
10         sum = 0,
11         evenOrOddIndicator = 0;
12
13     FILE *infile = NULL;
14     infile = fopen ("numbers.txt","r");
15
16     while (!feof (infile))
17     {
18         fscanf (infile,"%d",&num); //Get the number
19         //Is it a multiple of 7, 11 or 13?
20         isMultiple (num, &multipleIndicator7, &multipleIndicator11, &multipleIndicator13);
21
22         switch (multipleIndicator7)
23         {
24             case 1: printf ("%d is a multiple of 7\n",num);
25             ~

```

Issue Logging

Outside Lab:

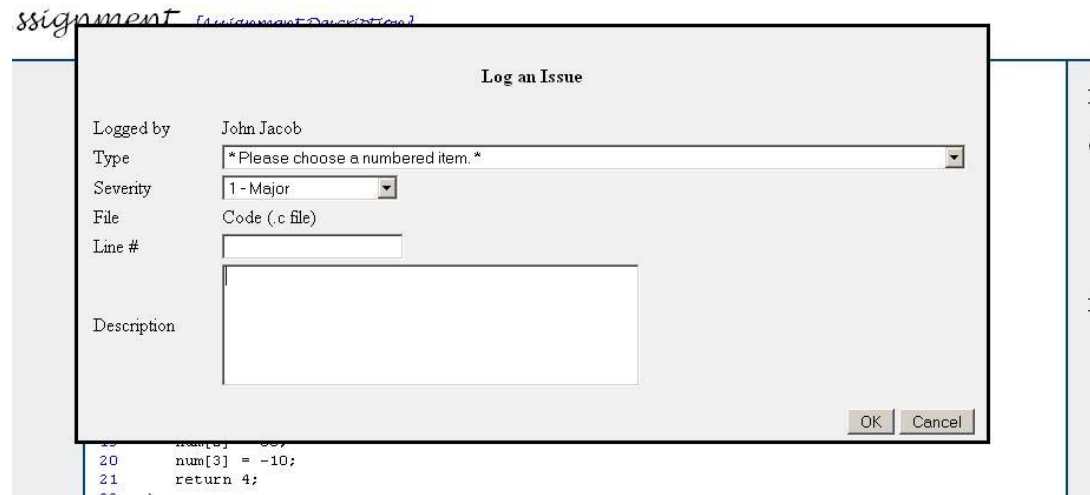
During Lab:

2. Logging an outside-lab issue on a particular line in his this assignment solution file:

- a. To log an outside-lab issue on the code, we click on the “Log an Issue” button, shown in Figure 17, under the “Outside Lab:” label. Alternatively,

we can also click the hyperlink associated with the line number, on which we want to log an issue. This will bring up a pop-up box for logging in the issue by entering its details (Figure 21).

Figure 21. Enhanced OSBLE interface to log an issue



b. Using the above interface, we can enter all the details related to an issue, like its type, severity, location and description. After entering all the information, we need to click on the “OK” button. The line on which we have logged an outside-lab issue is shown highlighted in yellow color (Figure 22).

Figure 22. Enhanced OSBLE interface showing highlighted lines of code for which issues have been logged – Yellow line has an outside-lab issue logged

```

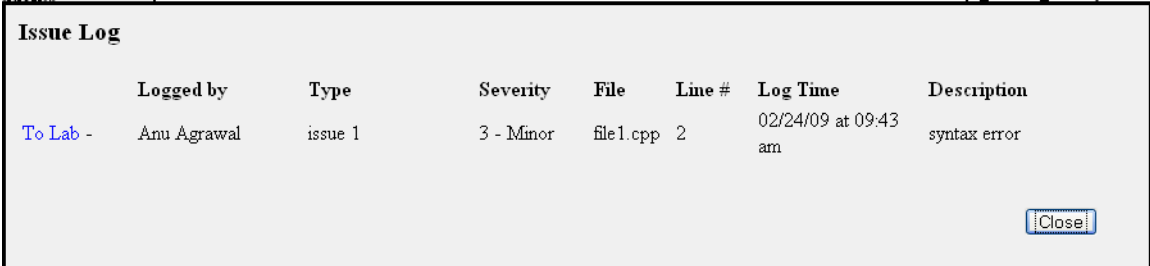
13
14 /// Description: Calculates the sum of an array of positive integers.
15 /// Input: pointer to integer array, ending in -1 to signify the end
16 /// of the array.
17 /// Output: sum of the array items up to the final -1.
18 int calculateSum(int* nums)
19 {
20     int i = 0;
21     int sum = 0;
22     while (nums[i] != 0)
23     {
24         i++;
25         sum += nums[i];

```

3. Transferring the issue logged outside-lab into the list of in-lab issues:

- a. To transfer an outside-lab issue to the list of in-lab issues, we first click on the “View issue Log” button, shown in Figure 20, under the “Outside Lab:” label. Alternatively, we can also click the hyperlink associated with any line of the code, for which an outside-lab issue has been logged. This will bring up a pop-up box displaying a list of all the outside lab issues logged (Figure 23).

Figure 23: Pop-up showing the list of all outside-lab issues logged



	Logged by	Type	Severity	File	Line #	Log Time	Description
To Lab -	Anu Agrawal	issue 1	3 - Minor	file1.cpp	2	02/24/09 at 09:43 am	syntax error

- b. Next we locate the issue, which is to be transferred and click on the “To Lab” hyperlink (as shown in Figure 23 above) associated with that issue. After transferring the issue, we simply click on the “Close” button. By performing these steps, we successfully transfer an issue from outside-lab to in-lab list.

To summarize, this chapter has discussed the enhancements made to the online tool OSBLE to support pedagogical code reviews. After completing the implementation of these enhancements during the Fall 2008 semester, we again implemented pedagogical code reviews in the CS1 course at WSU in the Spring of 2009. This time, the code reviews were supported by our online tool OSBLE and we collected and analyzed data to evaluate the effectiveness of our online tool OSBLE to support studio-based approach for computer science instruction. The following chapter presents an evaluation study of OSBLE conducted in Spring 2009.

CHAPTER SIX

IMPLEMENTATION OF OSBLE IN CS1 COURSE AND MAIN FINDINGS

As discussed in chapter 1, this research work is closely related to action research, which is conducted in natural setting and focuses more on improving processes rather than getting “Yes” or “No” answers to problems. Hence the primary techniques employed for data collection in this research include ethnographic field techniques. In this chapter, I first talk about ethnographic field techniques and then present the main findings for Spring 2009 semester.

Ethnographic Field Techniques

We primarily used Ethnographic Field Techniques for gathering data. As mentioned above, Action Research does not focus on getting a ‘Yes’ or ‘No’ answer to a question; rather, it focuses more on making observations as to how we can improve the process that we are studying as part of our research. In our case, the process is the Studio-based approach. Hence, we went beyond the boundaries of quantitative data analysis and focused on qualitative measures for making critical observations and finding interesting patterns in the process. Ethnographic field techniques are appropriate for this kind of study, as the study was conducted in field settings (regular labs of CS courses) as opposed to traditional research experiments conducted in a controlled lab setting.

Overall study for CPATH project relied on four ethnographic field techniques: a) videotaping, b) artifact collection, c) questionnaires, and d) interviews.

First, we employed *videotaping* in Spring 2008 and Spring 2009. We videotaped each team while they participated in pedagogical code reviews. In the Spring of 2008, the

recordings were done in the lab itself, with the camera focusing just on one team. We did not obtain acceptable audio quality in these recordings, as the voices of members of other teams interfered with the voices of the members of the team that was being recorded. Thus, in Spring of 2009, we did the recordings in a separate room, with only the team to be recorded present there. We made sure that the setup in the lab was similar to the setup in the lab. The post-hoc analysis of these videos pointed out some interesting results, which are discussed later in this chapter.

Second, we used *artifact collection* in all the semesters. In fall of 2007, we collected the “design crits” that the students prepared for fellow students who presented their work. In Spring 2008 and 2009, we collected the issue logs, which the teams prepared as part of the code review process. We subsequently evaluated students’ issue logs to determine the number and percentage of issues logged in each category and to explore patterns in logging issues.

Thirdly, we used *questionnaires* in all the semesters. We administered a written exit questionnaire containing open-ended questions that asked students to reflect on how they approached the activities they undertook in studio experiences, and what they thought of them. Students were required to fill in these questionnaires at the end of every studio-based lab session.

Finally, we used *interviewing* in all the semesters. Interviews were conducted with selected students at the end of each semester. These students represented the entire range of students in the class, as they were a good mix of all - high, medium and low scoring students. In order to pursue themes that emerged from our video recordings and exit questionnaires, I audiotaped and subsequently transcribed these interviews.

For my thesis work, I primarily used two of the ethnographic field techniques discussed above: a) Videotaping, and b) Artifact Collection.

In the following section, I discuss in brief the implementation of code-review sessions in a CS1 course at WSU in Spring 2009 semester, supported by online tool OSBLE. I then discuss in detail the main findings of our study during this semester.

Implementation of code reviews in CS1 course using OSBLE in Spring 2009

In 2009, we again implemented studio-based approach in a CS1 course by requiring students to participate in pedagogical code review sessions. The other details regarding the format in which code reviews were conducted in Spring 2009 semester were exactly same as the implementation of code reviews in CS1 course in Spring 2008 semester, discussed in detail in Chapter 4. The main difference this semester (Spring 2009), was the implementation and use of OSBLE in the course to support these code reviews (Please refer the Appendix for all the training materials used).

The class was divided into 4 sections consisting of roughly 20 students each. We implemented face-to-face code reviews supported by a limited version of OSBLE (which allowed the users to submit and view the code using OSBLE, but the users had to still log the ahead of time as well as in-lab issues on paper, without using OSBLE) in 2 of the 4 sections. These 2 sections will henceforth be referred as the “view-only-OSBLE” sections. In the other 2 sections we implemented face-to-face code reviews augmented by full version of OSBLE, which allowed the users to submit and view the code using OSBLE and also to log the ahead-of-time as well as in-lab issues online using OSBLE. These 2 sections will henceforth be referred as the “full-OSBLE” sections. Students (in

all 4 sections) were given training on the code review process. I collected and analyzed data (issue logs and video recordings of the review sessions) to measure the effectiveness of online tool OSBLE to support code reviews. In next section of this chapter, I present the main analysis of these data and the main findings for this semester.

Analysis of Video Recordings

Based on an analysis of the video corpus, we can make the following preliminary observations regarding the effectiveness of OSBLE system to facilitate face-to-face code reviews:

- *OSBLE improved the organization of code reviews.* One of the issues identified in Spring 2008 (lack of organization due to lot of paper print-outs) was overcome by using OSBLE. In Spring 2008 we observed that often students did not bring the print-outs of their assignment solutions to the lab, and so the sessions suffered as the students and moderators had to run at the last moment to get the print-outs. OSBLE overcame this issue by providing a central repository.
- *OSBLE facilitated easier access to issue logs.* Students could log issues through the on-line system, ahead of time, before attending face-to-face code reviews in lab. Also, students could easily view and integrate issues identified prior to the review into face-to-face review. OSBLE also provided all team members with permanent record of the review, which they can access for future reference.
- *OSBLE sped up the process of recording an issue in a code review.* We observed that the average time to record an issue using OSBLE was approximately about half the time required to record an issue with pen and paper. (This observation is

based on the analysis of half of the entire corpus of videos. We will be able to provide the exact figures after we have completed analysis of all the videos)

- *OSBLE helps students to highlight code while reading.* In code reviews, students ran OSBLE using a shared display, often using mouse hovering to indicate the code that they were currently focused on (see Figure 2 for a snapshot of this setup). This helped to focus everybody’s attention on the same segment of code, and kept everyone “on the same page.”
- *Students recorded issues collaboratively.* While recording the description of an issue, all team members participated in the process collaboratively. That is, they read aloud what the Recorder wrote, and also corrected the Recorder if they found a mistake. Table 2 below presents a small vignette from one of the video recordings, showing how a team of 4 students and a moderator logged an issue collaboratively (As the audio quality of the recordings was not good, the text of talk in Table 2, presents a close approximation of what the participants said).

Table 2: Vignette from a video recordings, showing how team logged issue collaboratively

Speaker	Talk
Student A	(while reading the text) s counter not equals null
Student B	i think some issue with notation
Moderator	could have used pointer notation throughout
Student C	line 25 and 27
Student D	should be in section 2
Student B	ya, it kind of looks like in section 2 or 1
Moderator	maybe this is in section 1 as well
Student B	(while logging the issue types “should use pointer notation, rather” and says) hmm...
Moderator	Array

- *Studio setting increased student participation.* Several videos showed that having small groups of students sitting around a common table and sharing a display

increased student participation and interest in the code review process, as illustrated in Figure 24.

Figure 24. Snapshot of the Team Code Review Set-Up with Shared Display



With the help of OSBLE, it also became possible for all the students to access the issues that had been logged even after the class for future references.

Analysis of “Ahead-of-time” reviews

In the Spring 2008 study, we did not require students to do ahead-of-time reviews, so, we were not able to use any data from that semester to compare paper-based code reviews with OSBLE-supported code reviews. The comparison that follows was done between the view-only-OSBLE and full-OSBLE sections in the CS1 class in spring 2009. Surprisingly, there were no “ahead-of-time” reviews done by any of the students in full-OSBLE section across all three code reviews. Whereas, there were three to five instances in the “view-only” sections where students came prepared to the code review labs with ahead-of-time issues logged on their outside-lab issue log sheet.

There were a few instances in which “view-only-OSBLE” section students forgot to bring their outside-lab issue log sheet to the lab, though they had filled it in. Two of these

students told their moderator that it would have been easier for them if these issue logs had been available online. This was quite an interesting observation, as the students of “view-only-OSBLE” section who did not have access to online issue logs indicated their desire to log issues online. Yet, students in “full-OSBLE” section who had this capability never used it.

Analysis of Issue Logs

We analyzed the issue logs, which code review teams had prepared as part of their studio-based activity to compare Spring 08 (paper-only) treatment with Spring 09 (Full OSBLE).

Table 3 and Table 4 present the number of issues logged by the three consistent groups in each of the three code reviews in the Spring 08 (paper-only) treatment and the two consistent groups in Spring 09 (Full OSBLE) treatment. The tables break the issues down further by the seven issue types described in Table 1 (p. 43).

Table 3. Categorization of inspection log issues by session (Spring 08 – paper-based)

#	Issue Category	Session 1	Session 2	Session 3
1	Meets assignment requirements	0 (0%)	0 (0%)	2 (7%)
2	Structure and design	9 (22%)	18 (38%)	3 (10%)
3	Documentation, standards & formatting	8 (20%)	5 (10%)	1 (3%)
4	Variables & constants	17 (41%)	13 (27%)	5 (17%)
5	Arithmetic operations	0 (0%)	1 (2%)	0 (0%)
6	Loops & branches	5 (12%)	6 (13%)	7 (24%)
7	Defensive programming	2 (5%)	5 (10%)	11 (38%)

Table 4. Categorization of inspection log issues by session (Spring 09 – full OSBLE)

#	Issue Category	Session 1	Session 2	Session 3
1	Meets assignment requirements	2 (7%)	4(12%)	13(38%)
2	Structure and design	5(18%)	13(39%)	7(21%)
3	Documentation, standards & formatting	5(19%)	2(6%)	0(0%)
4	Variables & constants	6(23%)	6(18%)	6(17%)
5	Arithmetic operations	0(0%)	1(3%)	0(0%)
6	Loops & branches	2(7%)	3(10%)	1(3%)
7	Defensive programming	7(26%)	4(12%)	7(21%)

According to a chi-square test, the distribution of issue types across the three code reviews differed significantly in both the Spring 08 (paper-only) treatment [$\chi^2(12, N = 117) = 33.2, p = 0.001$], and the Spring 09 (Full OSBLE) treatment [$\chi^2(12, N = 94) = 23.6, p = 0.023$]. However, the nature of the distributions of issues differed markedly between the two treatments.

For Spring 2008 (paper-based) condition, we can see that the number of issues in issue category 3 (Documentation Standards and Formatting) dropped gradually as the review sessions progressed, from 20 percent to 3 percent. Similar trend was seen in the percentage of category 4 issues (Variables and Constants), which also dropped from 41 percent in first review to 27 percent in second review and eventually 17 percent in third review. We can also see another trend, where the percentage of issues logged in issue category 6 (Loops and Branches) and issue category 7 (Defensive Programming), gradually increases as the reviews progress from 12 to 24 percent and 5 to 38 percent respectively.

Interestingly, the distribution of issue types across the three code reviews differed significantly for spring 2009 (Full OSBLE) semester as well. As we can see from Table 3, the percentage of issues logged in issue category 1 (Code meets assignment requirements), increased 7 percent to 39 percent as the sessions progressed. No such observation was made in Spring 2008. Also, the percentage of issues in category 3 (Documentation Standards and Formatting) dropped gradually as the review sessions progressed, from 19 percent to 0 percent. This drop in category 3 issues, as the sessions progressed was seen both in Spring 2008 and Spring 2009 sessions.

Table 5 presents a comparison of the mean number of issues logged per group in the Spring 08 (paper-only) and Spring 09 (OSBLE) offerings of CptS 121. This table further breaks the issues down by category, using the categories described in Table 1 (p. 43). It is important to note that, while the groups considered in this analysis were consistent across all three code reviews in the Spring 08 (paper-only) treatment, they were not consistent across all three code reviews in Spring 09 (OSBLE) treatment, owing to several student and moderator absences that occurred in the Spring of 09. Therefore, we have included the sample size of each specific review in the table. We found no significant differences ($df = 1$, $H = 1.76$, $p = 0.18$).

Table 5. Mean number of issues logged per group in each code review by treatment

Review #	Treatment	n	<i>Issue Category</i>							Total
			1	2	3	4	5	6	7	
Review 1	Paper-Only	3	0	9	8	17	0	5	2	41
	OSBLE	9	19	33	51	20	1	13	25	162
Review 2	Paper-Only	3	0	18	5	13	1	6	5	48
	OSBLE	7	12	22	11	13	1	21	7	87
Review 3	Paper-Only	3	2	3	1	7	0	8	7	28
	OSBLE	8	31	21	13	13	0	2	22	102

Due to the small sample sizes, we employed non-parametric Kruskal-Wallis ANOVAs to test for significant differences between the two treatments. To guard against Type-I error, we set the threshold alpha value in these tests to 0.00625 (0.05 divided by

eight). Table 6 presents the results of these Kruskal Wallis tests. As can be seen from the table, no statistically-reliable difference emerged. Results of our Kruskal Wallis tests show that there was statistically no difference in the number of issues logged in both the conditions (paper-based and OSBLE). There was neither any statistical difference in the total number of issues logged in each review session, nor there was any difference between the numbers of issues logged in each category, in both conditions, in any of the three review sessions.

Table 6. Statistical comparisons of the two treatments with respect to issues identified in the code reviews

Review #	Issue #	df	H	p-value
1	1	1	3.17	.0751
	2	1	0.00	1.000
	3	1	2.52	.1122
	4	1	4.95	.0260
	5	1	0.33	.5637
	6	1	0.15	.7003
	7	1	0.35	.5569
2	1	1	2.39	.1220
	2	1	2.26	.1325
	3	1	0.06	.8117
	4	1	2.53	.1119

	5	1	0.43	.5127
	6	1	1.69	.1930
	7	1	0.69	.4074
3	1	1	2.79	.0946
	2	1	1.39	.2378
	3	1	0.46	.4996
	4	1	0.91	.3413
	5	1	0.00	1.000
	6	1	6.13	0.0133
	7	1	0.04	.8325

Discussion

As is evident from the analysis of the video recordings, there were two clear benefits to augmenting the studio-sessions with OSBLE: improved organization of the studio session and the availability of a central repository for all the assignments posted as well as the issues logged by the students. A subtle benefit of OSBLE that emerged from this video analysis was that OSBLE helped in increasing students' concentration and hence their participation by making use of a single shared display. Also, we found episodes in the videos that indicated that the Reader often hovered the mouse over the text or selected a particular text while reading. This in particular helped in keeping everybody on the team focused on the same piece of code. Other than this, OSBLE helped in speeding up the recording process and was eco-friendly, obviating the need to print out multiple copies of studentss assignment solutions for the code reviews.

Although the video analysis provided some positive evidence of the effectiveness of OSBLE in supporting studio sessions, we did not get such promising results from our analysis of the issue logs. There were no significant differences in the total number of issues logged in the paper-based code review sessions and the OSBLE-supported code review sessions. There was only one significant difference found between the two conditions with respect to issues logged in the individual issue categories. This difference was in the number of Type 6 “Loops and Branches” issues logged in code review 2, which in itself is not sufficient to shed any light on the overall effectiveness of OSBLE.

As for the analysis of number of ahead-of-time issues logged in both conditions, the results were not promising in favor of OSBLE. I see two probable reasons as to why we did not get the results we expected. First, we may not have trained students sufficiently in the use of the OSBLE tool. Because of this, it is possible that the students were not confident enough to use the tool outside lab to do their reviews. Another possible reason could be that students were not given enough time to familiarize themselves with the tool. Indeed, during the entire semester, students only used OSBLE for three assignments.

Threat to Validity

In trying to reconcile the video analysis support for efficiency of OSBLE, I can identify one observation, which can be a potential threat to validity. The lab in which videos were recorded had a set-up with single shared display. This lab did not have provision for extra internet ports. But it had wireless internet connection, in case, students wanted to use their own laptops during the review. Whereas in the lab, where teams

conducted rest of their code reviews (which were not recorded), that lab was equipped with internet ports as well wireless internet connection.

Though, this difference in set-up, where one lab provided more ways of using individual laptops than the other one, might be a threat to validity, but there were two other contradictory observations made: a) None of the students was ever seen using the wireless connection in the video-recording lab, to use his individual laptop, instead of looking at the shared display, and b) Even though students had provision for using individual laptops in the other lab, they were often observed sharing laptops with each other. Based on these two observations, we can assume that this might not have been a major threat to validity.

Problems with Data Collection

Three different observations regarding the way in which we collected data might help explain the results we obtained. First, In Spring 2008 (paper-based) session, three out of four, or 75 percent of the teams were intact throughout the three code review sessions. This gave students in these teams an opportunity to become comfortable in working with each other and their moderator as a team, and also become comfortable with the entire code review process. This constant participation in the code review process gave students in these intact teams an opportunity to gradually start participating more actively in the review process as the sessions progressed. In contrast, there was a high attrition rate in the teams that participated in Spring 2009 (full-OSBLE) session. Out of the total nine teams, only two teams (22 percent) were intact through the three code review sessions. This high attrition rate in teams was due to the irregular attendance of students as well as moderators in spring 2009 session. At times, students were absent,

resulting in partial groups and at other times some of the moderators were absent, due to which the teams had to be merged, which changed the team formation completely. Thus, students in the spring 2009 session were mostly teamed up with different students and different moderator each time, due to which they never got a chance to become comfortable with their team members or with the entire review process.

Second, due to certain constraints, we were not always able to evaluate the performance of OSBLE against the appropriate condition: paper-based code reviews. As OSBLE was designed to augment face-to-face, paper-based reviews, it would be most fair to compare it against completely paper-based reviews. However, due to certain practical constraints, at places I compared full-OSBLE with a limited view-only version of OSBLE. In future, if we get a chance to compare OSBLE with paper-based reviews, it will be interesting to see how OSBLE performs with respect to student assignment resubmission grades. OSBLE certainly has the potential to perform better on this criterion, as it makes assignment solution of other students as well as the review logs accessible to the students accessible even after the lab is over, as opposed to paper-based reviews, where this is not a possibility.

We also collected some other artifacts, analysis of which could have thrown light on the effectiveness of OSBLE to support studio-sessions. We conducted pre- and post-tests in the beginning and end of each semester to measure students learning outcomes. Also, to measure student aptitude and perception, we asked the students to fill in pre- and post-survey in the beginning and end of the semester. Due to this practical constraint (of using view-only OSBLE instead of completely paper based code reviews for other 2 sections), we were not able to use these artifacts to analyze the impact of OSBLE in

enhancing studio-sessions. If we get an opportunity to compare OSBLE against paper-based studio sessions in future, we will certainly find some interesting positive results by analyzing these artifacts.

Another major issue was that we implemented OSBLE in CS1 courses for a limited time period. Students used OSBLE only for three assignments throughout the semester, which did not give them sufficient time to familiarize themselves with new software. By the time students became comfortable with the reviews, they were done with the three review sessions. This time constraint was certainly not a problem with paper-based reviews, as the students quickly became comfortable using pen and paper.

To summarize, in this chapter, I have presented the main findings of the OSBLE evaluation study we conducted during Spring 2009 semester. The chapter also discussed in detail some of the major problems related to data collection. If these problems had not been present, we would have been able to obtain a fuller evaluation of OSBLE's ability to support studio-based learning. In the next chapter, Chapter 7, I discuss the scope of future work related to OSBLE.

CHAPTER SEVEN

FUTURE WORK

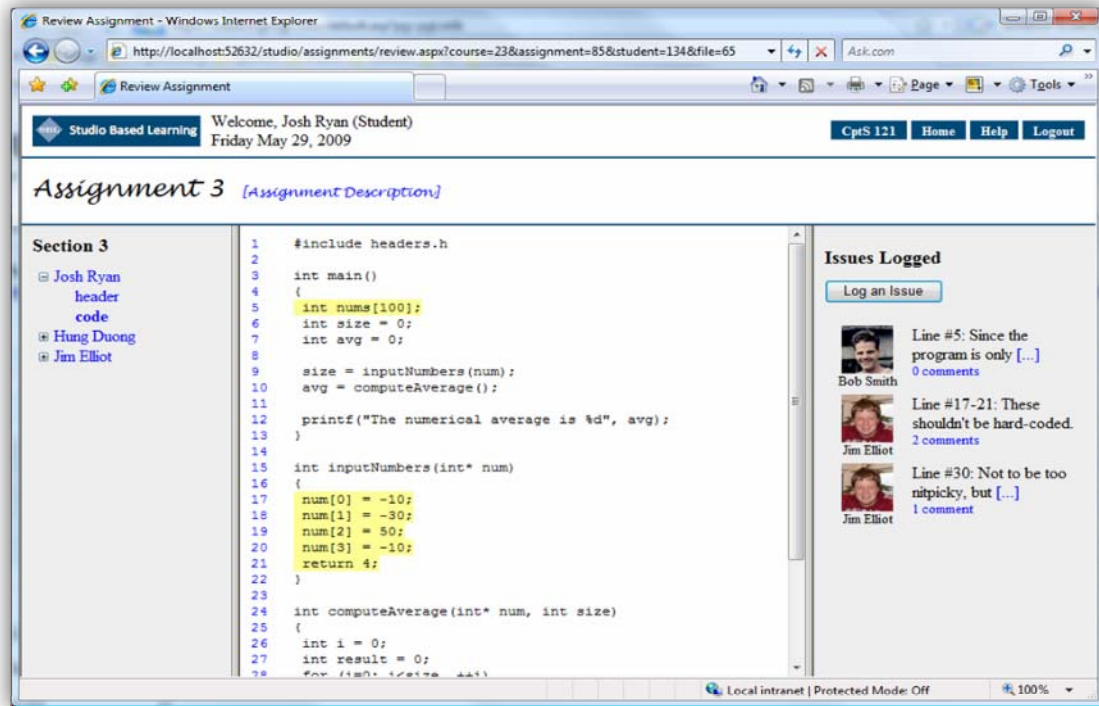
The results presented in this thesis have shed light on both the research questions originally established for this work. With respect to RQ1 (“How can we take advantage of an asynchronous online learning system to improve and tailor the studio based approach for teaching computer science”), we found evidence in our analysis of the video recordings that an asynchronous online tool can help in improving the organization of the studio sessions and also help in increasing student participation and collaboration. As discussed in previous chapter, there are other potential benefits to augmenting studio sessions with an online tool. Some of these potential benefits include improvement in assignment resubmission grades and increased online student participation from outside the lab.

With respect to RQ2 (“What might the design of such a system look like”), we have presented the OSBLE, a specific on-line peer review system that evolved through an iterative, user-centered design process. The current version of OSBLE can be considered as a reasonable starting point for the design and scope of such a system. By doing more studies with OSBLE by implementing it in other computer science courses, and possibly at other institutions, we will be able not only to improve the effectiveness of the OSBLE, but also to better understand its impact on learning computer science. In the remainder of this chapter, I outline directions for future research.

To increase overall student participation, to increase ownership in students' contributions, and to make the system more engaging and fun to work with, we would like to explore the possibility of enabling students to personalize their presence in OSBLE. For example, in the most recent version of OSBLE presently under

development, we allow students to add profile pictures. To increase students' perceived ownership in their on-line contributions, we show the profile pictures of students together with the comments and issues that they log, as illustrated in Figure 25.)

Figure 25: Snapshot of latest version of OSBLE with profile picture of students



One of the most important observations is that neither our online tool OSBLE, nor the way we organize pedagogical code reviews, provided motivated students to participate actively in “ahead-of-time” code reviews. Social psychology research shows that reciprocation is a basic norm of human society, and we need to provide people with appropriate rewards when we ask them to do something (Cheng & Vassileva, 2005). To incorporate this, in future we will explore the use of following incentive mechanisms and evaluating them: (1) provide a mechanism to rate the issues that have been logged, and (2) provide a mechanism of assigning stars/titles (e.g. “top reviewer”, “active reviewer”, “dormant reviewer” etc.) to individual students based on (a) the number of issues they

logged, (b) rating of the issues logged by them, and (c) their response to the issues logged for their code.

Research shows that the combination of reputation-based incentives and sending persuasive messages increases user participation and contribution as users tend to log in more frequently to the system (Harper, 2007). A study for rating movies online (Cheshire & Judd Antin, 2008, pp. 705-727) shows that reminding participants about their uniqueness and importance of their feedback through email messages helped in improving the response. In the future, we plan to incorporate this into our system, by sending automated, personalized emails to a student (through OSBLE) when (a) the instructor posts a new assignment, (b) a student on his/her team submits their assignment, and (c) students the TA, or the instructor log issues or post comments on his/her assignment. We will collect data and conduct research studies to find out the most appropriate content and frequency of these kinds of reminder messages sent through email.

We understand that using e-mail as the main interface for information awareness can result in an overloaded inbox (Simpson, Burmeister, Boykiw, & Zhu, 2003). To avoid this problem, we plan to send only personal and team-centric updates through email notifications. We will provide users with more detailed updates for all the activities going on at the class-level, when a student logs in to the system. Every time the student logs in, he or she will be given class-level updates for (a) links to all student assignments solutions posted since he or she last logged in, (b) a list of all the issues logged by any student in the class, and (c) information about who else is logged in to the system at the same time. In the future, we would like to explore the potential for peripheral awareness

tools like “Sideshow” and integrate them into OSBLE, in order to provide peripheral information without overloading or distracting its users (Simpson et al., 2003).

Another major challenge is to increase the interest of students to participate in pedagogical code reviews, make them realize the importance of participating in such reviews, and improve the entire format of these reviews to make them more interesting and less redundant. We propose to carry out the following steps in order to overcome this challenge: (a) provide detailed training to students in the beginning of the semester, introducing them to pedagogical code reviews and their importance, and (b) train the moderators on soft skills, including how they can increase student’s interest and participation in the review process.

To summarize, the research presented in this thesis has highlighted the potential of OSBLE as an online tool to support and augment studio-based sessions for computer science education. This research has evaluated the current version of OSBLE and also pointed out areas of improvement in data collection and analysis strategies to be considered while evaluating OSBLE in the future. This work also has pointed out areas in which OSBLE can be improved, and has proposed specific features, which, if added to OSBLE, hold promise in improving its potential to serve as a successful tool to support studio-based learning for computing education.

BIBLIOGRAPHY

action research. (n.d.). Retrieved July 22, 2009, from

http://carbon.cudenver.edu/~mryder/itc/act_res.html.

Agrawal, A., & Hundhausen, C. D. (2008). The design of an asynchronous web-based project review system to support studio-based learning in computing education. In *Proceedings of the 2008 IEEE Symposium on Visual Languages and Human-Centric Computing - Volume 00* (pp. 254-255). IEEE Computer Society.

Retrieved July 22, 2009, from

<http://portal.acm.org/citation.cfm?id=1549823.1550011&coll=GUIDE&dl=GUIDE&CFID=44949541&CFTOKEN=33727890>.

Anewalt, K. (2005). Using peer review as a vehicle for communication skill development and active learning. *Journal of Computing Sciences in Colleges*, 21(2), 148–155.

Bali, M., & Ramadan, A. R. (2007). Using rubrics and Content Analysis for Evaluating Online Discussion: A Case Study from an Environmental Course. *Journal of Asynchronous Learning Networks*, 11(4).

Beaudin, B. P. (1999). Keeping online asynchronous discussions on topic. *Journal of Asynchronous Learning Networks*, 3(2), 41–53.

Bhattacharya, M. (1999). A study of asynchronous and synchronous discussion on cognitive maps in a distributed learning environment. In *Proceedings of WebNet World Conference* (pp. 24–30).

Birnholtz, J. P., Finholt, T. A., Horn, D. B., & Bae, S. J. (2005). Grounding needs: achieving common ground via lightweight chat in large, distributed, ad-hoc groups. In *Proceedings of the SIGCHI conference on Human factors in computing*

- systems* (pp. 21-30). Portland, Oregon, USA: ACM. doi:
10.1145/1054972.1054976.
- Black, A. (2005). The use of asynchronous discussion: Creating a text of talk.
Contemporary issues in Technology and teacher education, 5(1), 5–24.
- Blackboard Home. (n.d.). Retrieved July 22, 2009, from <http://www.blackboard.com/>.
- Boyer, E. L., & Mitgang, L. D. (1996). *Building Community: A New Future for
Architecture Education and Practice. A Special Report*. California Princeton
Fulfillment Services; 1445 Lower Ferry Road, Ewing, NJ 08618 (\$15 plus \$3
shipping).
- Burgess, L. A. (2003). WebCT as an e-learning tool: A study of technology students'
perceptions. *Journal of technology education*, 15(1), 6–15.
- Cheng, R., & Vassileva, J. (2005). User motivation and persuasion strategy for peer-to-
peer communities. In *System Sciences, 2005. HICSS'05. Proceedings of the 38th
Annual Hawaii International Conference on* (pp. 193a–193a).
- Cheshire, C., & Judd Antin. (2008). The Social Psychological Effects of Feedback on the
Production of Internet Information Pools. *Journal of Computer-Mediated
Communication*, 13(3), 705-727. doi: 10.1111/j.1083-6101.2008.00416.x.
- Davidson-Shivers, G. V., Muilenburg, L. Y., & Tanner, E. J. (2002). How do students
participate in synchronous and asynchronous online discussions? *Journal of
Educational Computing Research*, 25(4), 351–366.
- Davis, J. R., & Huttenlocher, D. P. (1995). Shared annotation for cooperative learning. In
*The first international conference on Computer support for collaborative learning
table of contents* (pp. 84–88).

- Elliott, J. (1991). *Action research for educational change*. Open University Press Philadelphia.
- Freeman, M. A., & Capper, J. M. (1998). An anonymous asynchronous web-based role play. In *ASCILITE* (Vol. 98).
- Gehring, E. F. (2001). Electronic peer review and peer grading in computer-science courses. *ACM SIGCSE Bulletin*, 33(1), 139–143.
- Gehring, E. F., Chinn, D. D., Pérez-Quinones, M. A., & Ardis, M. A. (2005). Using peer review in teaching computing. In *Proceedings of the 36th SIGCSE technical symposium on Computer science education* (pp. 321–322).
- Gilb, T., Graham, D., & Finzi, S. (1993). *Software inspection*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.
- Harper, F. M. (2007). Encouraging Contributions to Online Communities with Personalization and Incentives. In *Proceedings of the 11th international conference on User Modeling* (pp. 460-464). Corfu, Greece: Springer-Verlag. Retrieved July 22, 2009, from <http://portal.acm.org/citation.cfm?id=1419416.1419492&coll=GUIDE&dl=GUIDE&CFID=44949541&CFTOKEN=33727890>.
- Hübscher-Younger, T., & Narayanan, N. H. (2003). Constructive and collaborative learning of algorithms. In *Proceedings of the 34th SIGCSE technical symposium on Computer science education* (pp. 6-10). Reno, Nevada, USA: ACM. doi: 10.1145/611892.611919.
- Hundhausen, C., Agrawal, A., Fairbrother, D., & Trevisan, M. (2009). Integrating pedagogical code reviews into a CS 1 course: an empirical study. In *Proceedings*

- of the 40th ACM technical symposium on Computer science education* (pp. 291–295).
- Hundhausen, C. D., Narayanan, N. H., & Crosby, M. E. (2008). Exploring studio-based instructional models for computing education.
- Kern, V. M., Saraiva, L. M., & dos Santos Pacheco, R. C. (2003). Peer review in education: Promoting collaboration, written expression, critical thinking, and professional responsibility. *Education and Information Technologies*, 8(1), 37–46.
- Lynch, K., Carbone, A., Arnott, D., & Jamieson, P. (2002). A studio-based approach to teaching information technology. In *Proceedings of the Seventh world conference on computers in education conference on Computers in education: Australian topics - Volume 8* (pp. 75-79). Copenhagen, Denmark: Australian Computer Society, Inc. Retrieved July 22, 2009, from <http://portal.acm.org/citation.cfm?id=820060.820075&coll=GUIDE&dl=GUIDE&CFID=44949541&CFTOKEN=33727890>.
- O'Brien, R. (1998). An overview of the methodological approach of action research. *Unpublished paper to Professor Joan Cherry, Course LIS3005Y, Faculty of Information Studies, University of Toronto. April, 17.*
- Preece, J., Rogers, Y., & Sharp, H. (2001). *Beyond Interaction Design: Beyond Human-Computer Interaction*. John Wiley & Sons, Inc. New York, NY, USA.
- Romiszowski, A. J. (1995). Use of hypermedia and telecommunications for case-study discussions in distance education. *Open and Distance Learning Today, New York, Routledge*, 164–172.

- Seltzer, W. (2000). Annotation engine (Technical report). Cambridge, MA: Harvard University. *Berkman Center for Internet & Society*.
- Simpson, M., Burmeister, J., Boykiw, A., & Zhu, J. (2003). Successful studio-based real-world projects in IT education. In *Proceedings of the fifth Australasian conference on Computing education - Volume 20* (pp. 41-51). Adelaide, Australia: Australian Computer Society, Inc. Retrieved July 22, 2009, from <http://portal.acm.org/citation.cfm?id=858403.858409&coll=GUIDE&dl=GUIDE&CFID=44949541&CFTOKEN=33727890>.
- Smith, M., & Doyle, M. (2007). Action Research: Action Research: Action Research: Action Research: Action Research: Traditions and P raditions and P raditions and P raditions and P raditions and Perspectives erspectives erspectives erspectives. *Teaching & Learning*, 4(2).
- Suthers, D. (2001). Collaborative representations: Supporting face to face and online knowledge-building discourse. In *PROCEEDINGS OF THE ANNUAL HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES* (pp. 99–99).
- Suthers, D., & Xu, J. (2002). Kukakuka: An online environment for artifact-centered discourse. In *Education Track of the Eleventh World Wide Web Conference (WWW 2002)* (pp. 472–480).
- Takeda, T., & Suthers, D. (2002). Online workspaces for annotation and discussion of documents. In *Computers in Education, 2002. Proceedings. International Conference on* (pp. 1294–1298).
- Thurmond, V. A. (n.d.). The Point of Triangulation. *Journal of Nursing Scholarship*, 33(3), 253.

- Trytten, D. A. (2005). A design for team peer code review. In *Proceedings of the 36th SIGCSE technical symposium on Computer science education* (pp. 455–459).
- Wiegers, K. E. (1995). Improving quality through software inspections. *Software Development*, 55–64.
- Yee, K. P. (1998). *CritLink: Better Hyperlinks for the WWW, 1998*.

APPENDIX

Online Studio-Based Learning Environment (OSBLE) Tutorial for Moderators

“Full” Version Used in Wednesday Sections
(last modified March 8, 2009)

Introduction

This document introduces you to the Online Studio-Based Learning Environment (OSBLE, pronounced “Ohs-Bull”), a web-based system specifically designed to support the three code reviews in which you participate this semester. This tutorial will walk you through the key tasks you will need to perform with the system:

- Accessing and logging in to the system (Tasks 1 and 2 in this tutorial)
- Viewing an assignment to be reviewed (Task 3)
- Viewing the code solutions of a student for a given assignment (Task 4)
- Starting an in-lab review (Task 5)
- Logging issues with a team member’s assignment solution within the in-lab review (Tasks 6 and 7)
- Finalizing an in-lab review (Task 8)

Please bear in mind that, while OSBLE has undergone a reasonable amount of testing, it is a research prototype, not a commercial system. As such, it is not as robust and stable as a commercial system. If you encounter unexpected bugs or problems with the system, please report them immediately by e-mail to Kyle Ryan (kdryan@wsu.edu), who will try to remedy the problem as quickly as possible.

Task 1: Accessing the OSBLE System

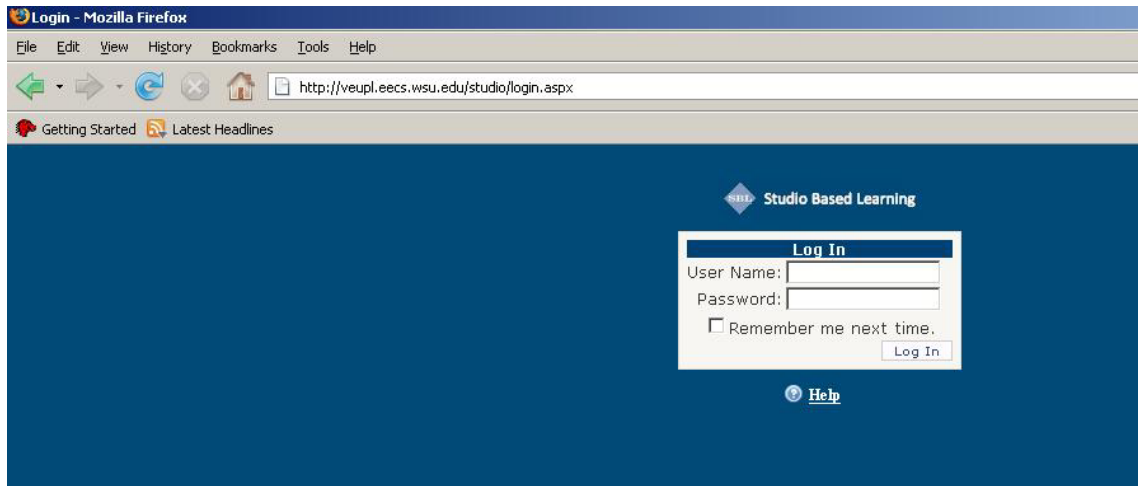
You can access and log in to the OSBLE system via the web.

Steps:

- Open a web browser. (We *highly* recommend using latest version of Firefox.)
- Type the following URL into the address bar of your browser:

<http://veupl.eecs.wsu.edu/studio/>

You will see the following login screen:



Task 2: Log in to the system first time

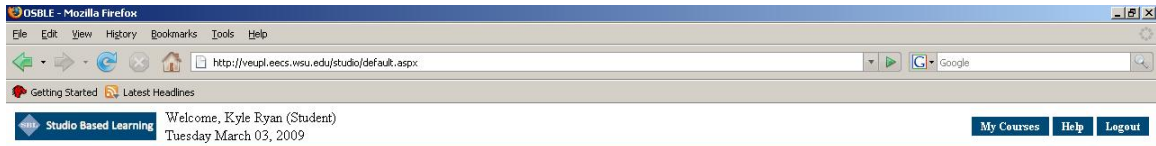
When you log in to OSBLE for the first time, you will use your Email address for both username and password. You will then be asked to change your username and password to something more memorable.

Steps:

- From the login screen, enter your e-mail address as the username.
- Enter your e-mail address as the password.

You will be taken to the following screen:

- Enter your preferred email address (it will become your username from this point forward).
- Enter your new password.
- Press “Submit.”
- You will be taken to the following screen:



<p>My Courses</p> <p>CptS 121: Program Design and Development</p>
--

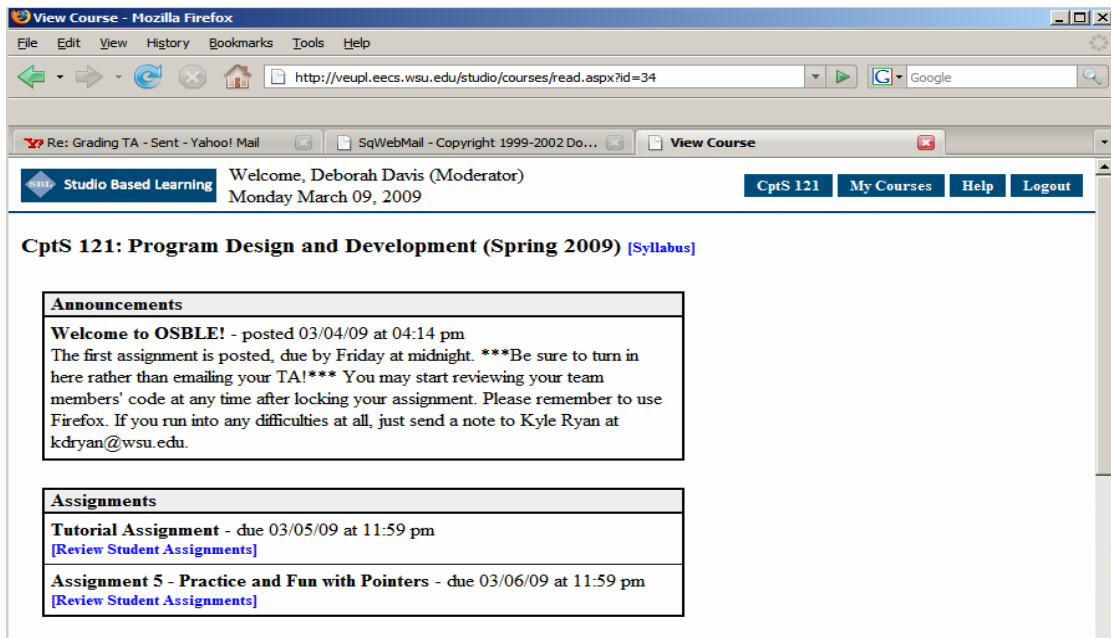
- Press “Logout” in the top-right corner.
- You will be taken back to the login screen. Here you can use the email address that you provided as the username and the new password that you provided for logging in to the system.

Task 3: View course and assignment

Once you log in to OSBLE, a list of courses for which you serve as a moderator appears. You can click on a particular course to view course announcements for that course (posted by an instructor or TA), and to access assignments on which you will perform code reviews in lab.

Steps:

- Log in to the system.
- Choose course “CptS 121 – Program Design and Development.”
- You will be taken to the following screen, which lists the assignments on which you will perform code reviews:

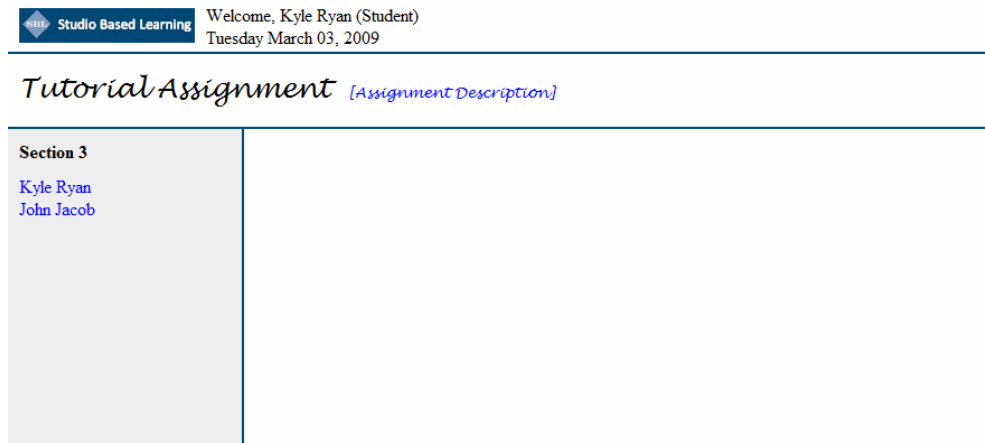


Task 4: View code of a student's assignment

Within an in-lab code review, you and your team will use OSBLE to view the solution code of the team members within each review that you perform. To view a team member's code, follow these steps:

Steps:

- Click on the “Review Student Assignments” link associated with the assignment for which you would like to review others' solutions.
- You will be taken to screen that looks similar to the following:



The screenshot shows the OSBLE interface. At the top left, there is a logo for "Studio Based Learning". To its right, the text reads "Welcome, Kyle Ryan (Student)" and "Tuesday March 03, 2009". Below this is a horizontal line. Underneath the line, the text "Tutorial Assignment" is displayed in a large, italicized font, with a smaller link "[Assignment Description]" to its right. Below this is another horizontal line. On the left side, there is a vertical panel with a light gray background. At the top of this panel is the text "Section 3". Below it are the names "Kyle Ryan" and "John Jacob" listed vertically.

- Click on the name of the student whose code you want to review. To access the names of the students on your review team, click the “List of Review Teams” link in the left panel on the top of list of all students.



The screenshot shows a vertical panel with a light gray background. At the top is the text "Section 3". Below it is a link "[List of Review Teams]". Below the link are the names "Daniel Cox", "Joshua Hardy", and "David Hawbaker" listed vertically.

- Once you select a student, the uploaded file names will appear under that student's name:

All Students

[\[List of Review Teams\]](#)

Robert Cardenas

- Computations - Header
- Computations - Driver
- Computations - Functions
- MathematicalModel - Header
- MathematicalModel - Driver
- MathematicalModel - Functions

Christina Ells
Kristopher Felt

- Select a student and solution file to review.
- The code of the file will appear in the middle panel, along with line numbers, as in the following screen shot:

```

1  #include "functions.h"
2
3  int main (void)
4  {
5      int num = 0,
6          primeIndicator = 1,
7          multipleIndicator7 = 0,
8          multipleIndicator11 = 0,
9          multipleIndicator13 = 0,
10         sum = 0,
11         evenOrOddIndicator = 0;
12
13     FILE *infile = NULL;
14     infile = fopen ("numbers.txt","r");
15
16     while (!feof (infile))
17     {
18         fscanf (infile,"%d",&num); //Get the number
19         //Is it a multiple of 7, 11 or 13?
20         isMultiple (num, &multipleIndicator7, &multipleIndicator11, &multipleIndicator13);
21
22         switch (multipleIndicator7)
23         {
24             case 1: printf ("%d is a multiple of 7\n",num);
25             ~

```

Issue Logging

Outside Lab:

[Log an Issue](#)

[View Issue Log](#)

During Lab:

[Start Inspection](#)

[Log an Issue](#)

[View Issue Log](#)

[Finalize Inspection](#)

Task 5: Start a Review

As team moderator, your responsibility is to lead the review of each team member's code. Each review will take place within OSBLE, and you are the only person (as moderator) who is allowed to start an in-lab review in OSBLE. Once a review has been started in OSBLE, the team's Recorder is able to add issues to the Team Issue Log. To start an in-lab review, follow these steps:

Steps:

- Click the “Start Inspection” button in the right panel. The following pop-up appears:

- Select the Reader, Inspector and Recorder from the list of students in the dropdown.

- The person you have assigned to be the **Recorder** will be able to add issues to the In-lab Issue Log. Others performing the review will be able to view the in-lab issue log.

Task 6: Log issues in the Team Issue Log within an in-lab review

As team moderator, your responsibility is to lead the review of each team member’s code. At the start of each review, you will lead the team in a line-by-line review of the code solution. Remember that you will strategically choose a starting point for this line-by-line review based on (a) what sections of code have been reviewed in prior reviews (so as to minimize redundancy), and (b) what parts of the code seem particularly prone to having issues (which could be determined either by peeking at the outside-of-lab review log for each team member ahead of time, or by anticipating where students are likely to have problems.) As issues are identified by your team during the code review, have the *Recorder* add them to the Team Review Log. Here, there are two possibilities:

1. *The team identifies an issue that an individual team member also added to the outside-of-lab review log.* Lines that are associated with issues that were logged by students outside of lab will be highlighted in yellow; however, issues logged outside of lab that pertain to a range of lines, or to an entire function, will not be highlighted, since range highlighting is not presently supported by OSBLE. When your team identifies an issue, consult the Outside-of-Lab log to see if the issue also appears there. If so, adding the issue to the In-Lab Review Log can be achieved by transferring the issue from the Outside-of-Lab Review Log to the In-Lab Review Log. To do this, have the **Recorder** perform the following steps:

Steps:

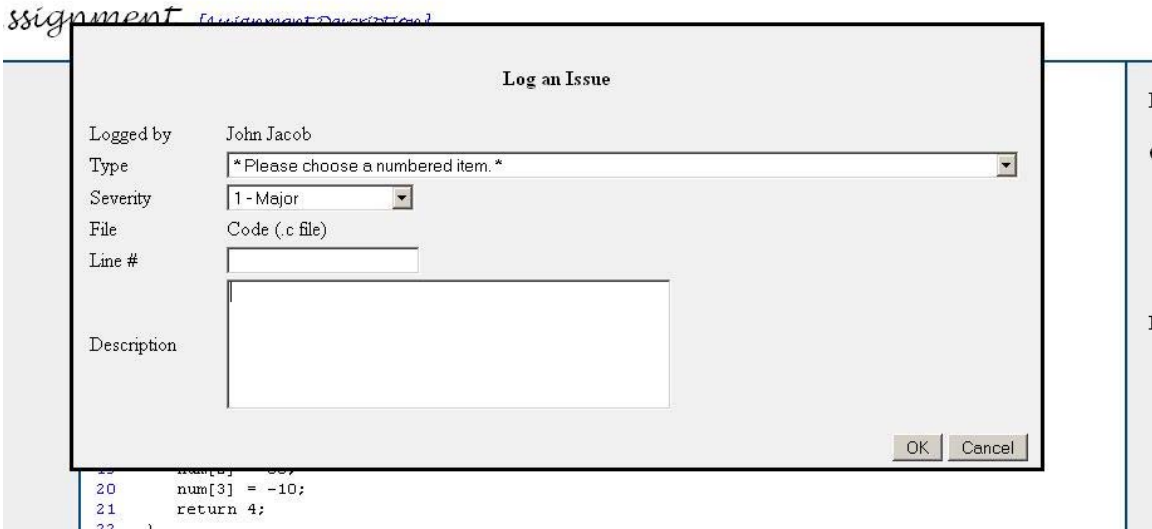
- Open the Outside-of-lab Issue Log by clicking on the “View Issue Log” button under the “Outside of Lab” header. A pop-up like the following will appear:

Logged by	Type	Severity	File	Line #	Description
Accept in Lab - Mod Mod Delete	1.3. ("Computations" project) main function implemented correctly?	1 - Major	Code (.c file)	20	test

- Find the issue that matches the one just identified by the team.
 - Click on the “Accept in Lab” button to the left of the issue.
 - The issue will then be automatically transferred to the Team Issue Log. If the issue pertains to a single line, that line will be highlighted in Orange to indicate that the issue was identified by the team, as opposed to by an individual.
 - Right now an issue that is transferred to the Team Issue log cannot be modified. If the team wants to modify something, the **Recorder** needs to create a new issue.
2. *The team identifies a new issue that was not identified by an individual team member outside of lab.* If the issue you have identified as a team is not similar to one of the issues logged outside of lab, have the **Recorder** add the new issue to the Team Issue Log as follows:

Steps:

- Click on the line number at which the issue was found. Alternatively, click on the “Log an Issue” button below the “In Lab” header on the left-hand pane.
- The following pop-up dialog box will appear:



- Note that you can drag this dialog around the screen if you need to look at the code underneath it.
- When you identify an issue, log the issue in a new line as follows.
 - Identify the type of issue from the code review checklist.
 - Specify whether the issue is Major (1), Moderate (2), or Minor (3).
 - Specify the location of the issue (a specific line number, function name, or range of line numbers).
 - Describe the issue in enough detail for the author to address the issue.
- Click “OK” to log the issue.
- When you log issues that pertain to single lines of code, those lines are automatically highlighted in orange to underscore that team-identified issues are associated with those lines, as in the following screenshot:

```

13
14 /// Description: Calculates the sum of an array of positive integers.
15 /// Input: pointer to integer array, ending in -1 to signify the end
16 /// of the array.
17 /// Output: sum of the array items up to the final -1.
18 int calculateSum(int* nums)
19 {
20     int i = 0;
21     int sum = 0;
22     while (nums[i] != 0)
23     {
24         i++;
25         sum += nums[i];

```

- If, however, an issue pertains to multiple lines or entire functions, the present version of OSBLE is unable to highlight the lines to which that issue pertains.
- You may click on a highlighted line to bring up the corresponding issue log.
- At any point, you can delete an issue that you have logged by viewing the issue log and clicking on the “Delete” button to the left of the issue.

Task 7: Transfer any remaining issues from the Outside-of-Lab Issue Log to the Team Issue Log

Be sure to stop the line-by-line review of each team member's solution at least **10** minutes prior to the end of the time allotted for the review (35 minutes are allotted for each review). This will allow your team to view the Outside-of-lab Review Log as a team, in order to see if any of the issues that individual members logged prior to the lab either were missed by the team, or simply were outside of the scope of the in-lab review. Have the team discuss each issue in the Outside-of-Lab Review Log that is not present in the Team Review Log. If the team decides that the issue should be included in the Team Review Log, have the **Recorder** transfer the issue to the Team Issue Log by following these steps:

Steps:

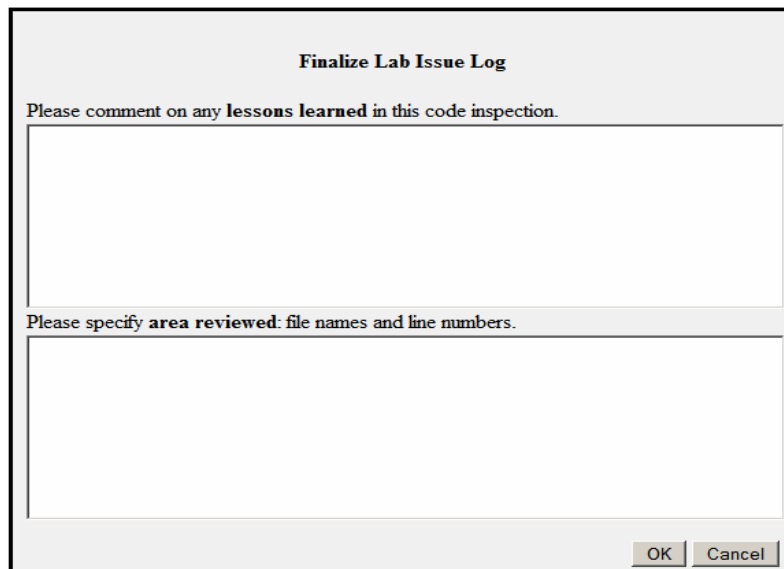
- Open the Outside-of-lab Issue Log.
- Find the issue that the team decided should be transferred to the Team Issue Log.
- Click on the "Accept in Lab" button to the left of the issue.
- The issue will be automatically transferred to the Team Issue Log.

Task 8: Finalize the Review

Once the team completes the review of a given team member's code solution, or the allotted time is up, you need to have the Recorder "finalize" the Review. This step tells OSBLE that the code review is complete, and it locks the Team Review Log so that no more issues can be logged. Have the Recorder follow these steps in order to finalize a review:

Steps:

- Click on the "Finalize Review" button in the right panel. The following pop-up appears:



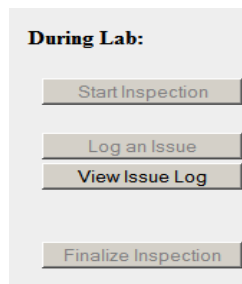
Finalize Lab Issue Log

Please comment on any **lessons learned** in this code inspection.

Please specify **area reviewed**: file names and line numbers.

OK Cancel

- Based on the team discussion, have the recorder fill in the sections labeled “lessons learned.”
- Ask the **Reader** to tell the **Recorder** exactly which files and line ranges (e.g., “Computations Header: Lines 1-20”) were covered in the Review. The **Recorder** should note these in the “area reviewed” text box, one line range per line.
- Click the “OK” button to finalize the review.
- Once the Issue Log has been finalized, all In-Lab Review buttons (except “View Issue Log”) will become disabled, indicating that the In-lab Review has been completed.



Online Studio-Based Learning Environment (OSBLE) Tutorial for Moderators

”View-Only” Version Used in Tuesday and Thursday Sections

(last modified March 8, 2009)

Introduction

This document introduces you to the Online Studio-Based Learning Environment (OSBLE, pronounced “Ohs-Bull”), a web-based system specifically designed to support the three code reviews in which you participate this semester. This tutorial will walk you through the key tasks you will need to perform with the system:

- Accessing and logging in to the system (Tasks 1 and 2 in this tutorial)
- Viewing an assignment to be reviewed (Task 3)
- Viewing the code solutions of a student for a given assignment (Task 4)
- Starting an in-lab review (Task 5)
- Logging issues with a team member’s assignment solution within the in-lab review (Tasks 6 and 7)
- Finalizing an in-lab review (Task 8)

Please bear in mind that, while OSBLE has undergone a reasonable amount of testing, it is a research prototype, not a commercial system. As such, it is not as robust and stable as a commercial system. If you encounter unexpected bugs or problems with the system, please report them immediately by e-mail to Kyle Ryan (kdryan@wsu.edu), who will try to remedy the problem as quickly as possible.

Task 1: Accessing the OSBLE System

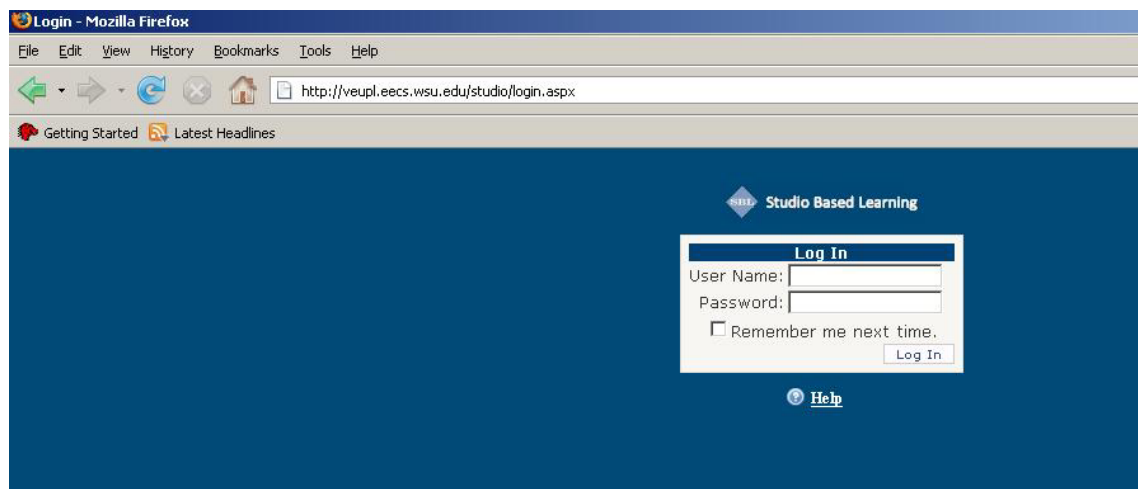
You can access and log in to the OSBLE system via the web.

Steps:

- Open a web browser. (We *highly* recommend using latest version of Firefox.)
- Type the following URL into the address bar of your browser:

<http://veupl.eecs.wsu.edu/studio/>

You will see the following login screen:



Task 2: Log in to the system first time

When you log in to OSBLE for the first time, you will use your Email address for both username and password. You will then be asked to change your username and password to something more memorable.

Steps:

- From the login screen, enter your e-mail address as the username.
- Enter your e-mail address as the password.

You will be taken to the following screen:

Studio Based Learning

Please enter your login information.

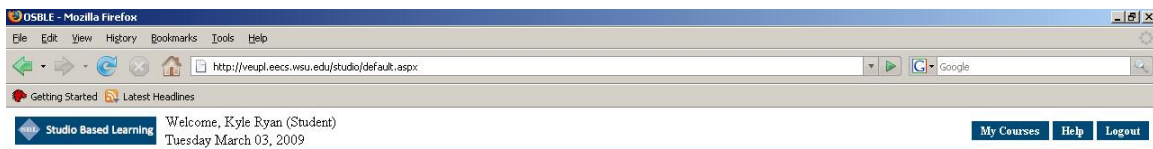
Email Address ← This is your username!

Confirm Email Address

Password

Confirm Password

- Enter your preferred email address (it will become your username from this point forward).
- Enter your new password.
- Press “Submit.”
- You will be taken to the following screen:



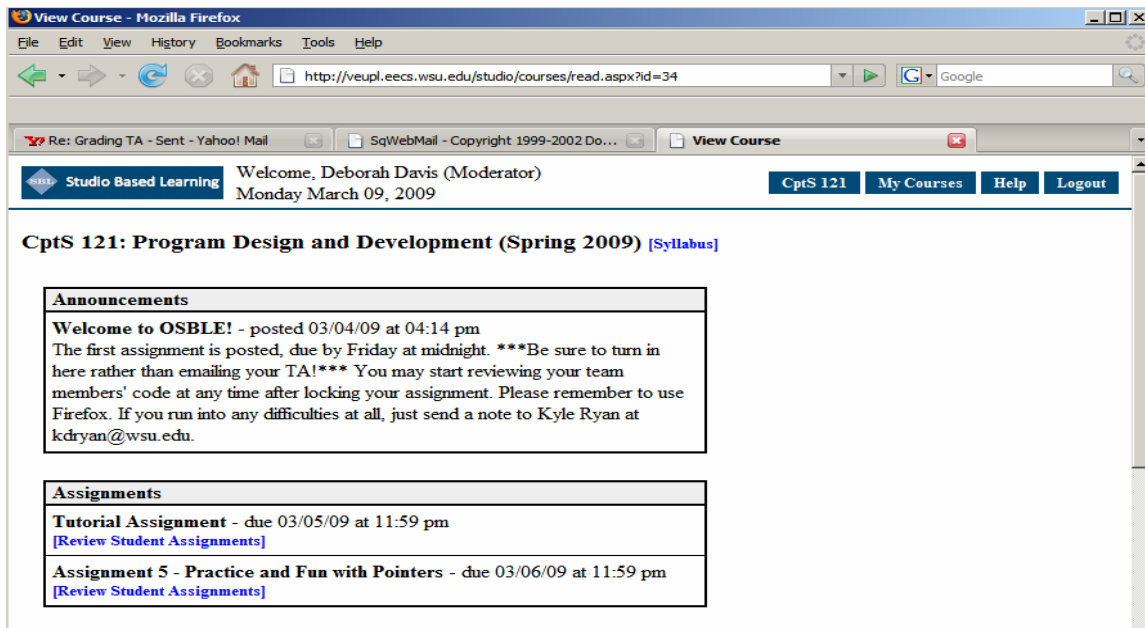
- Press “Logout” in the top-right corner.
- You will be taken back to the login screen. Here you can use the email address that you provided as the username and the new password that you provided for logging in to the system.

Task 3: View course and assignment

Once you log in to OSBLE, a list of courses for which you serve as a moderator appears. You can click on a particular course to view course announcements for that course (posted by an instructor or TA), and to access assignments on which you will perform code reviews in lab.

Steps:

- Log in to the system.
- Choose course “CptS 121 – Program Design and Development.”
- You will be taken to the following screen, which lists the assignments on which you will perform code reviews:

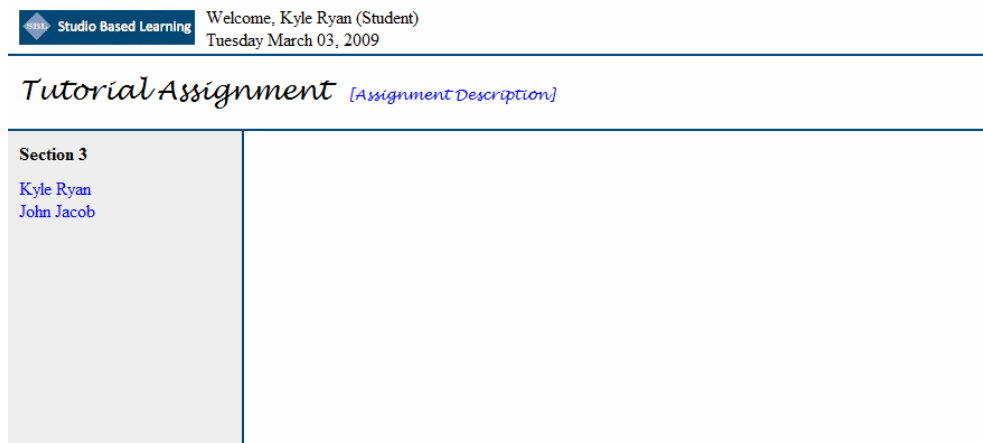


Task 4: View code of a student's assignment

Within an in-lab code review, you and your team will use OSBLE to view the solution code of the team members within each review that you perform. To view a team member's code, follow these steps:

Steps:

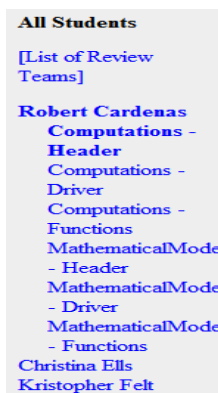
- Click on the “Review Student Assignments” link associated with the assignment for which you would like to review others’ solutions.
- You will be taken to screen that looks similar to the following:



- Click on the name of the student whose code you want to review. To access the names of the students on your review team, click the “List of Review Teams” link in the left panel on the top of list of all students.



- Once you select a student, the uploaded file names will appear under that student’s name:



- Select a student and solution file to review.
- The code of the file will appear in the middle panel, along with line numbers, as in the following screen shot:

```

1  /*
2  * Programmer: Robert Cardenas
3  * Assignment: PA5
4  */
5
6
7
8  #include "computations.h"
9
10 int main()
11 {
12     int num = 0;
13
14     FILE *infile = NULL;
15     infile = fopen("numbers.txt", "r");
16
17     while (!feof(infile))
18     {
19         read_int(infile, &num);
20         printf("Integer: %d", num);
21     }
22     return;
23 }

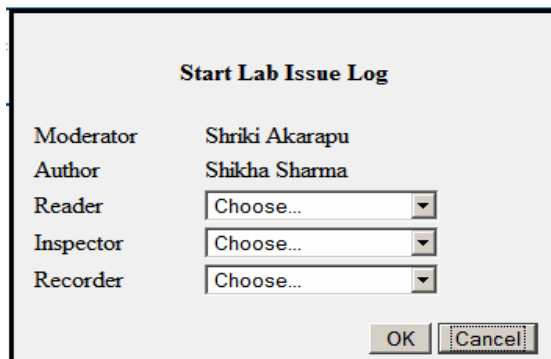
```

Task 5: Start a Review

As team *Moderator*, your responsibility is to lead the review of each team member's code. The team should use OSBLE to view the code under view. For research purposes, OSBLE will log the start and end time of each review. In order for it to do this, you need to tell OSBLE when the review starts and ends. To tell OSBLE that the in-class review is starting, follow these steps:

Steps:

- Click the “Start Inspection” button in the right panel. The following pop-up appears:

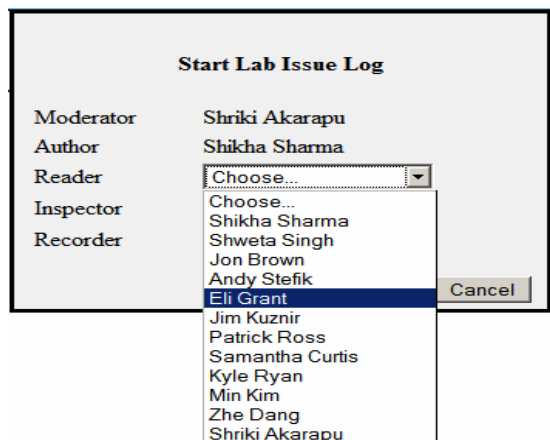


Start Lab Issue Log

Moderator Shriki Akarapu
Author Shikha Sharma
Reader Choose...
Inspector Choose...
Recorder Choose...

OK Cancel

- Select the Reader, Inspector and Recorder from the list of students in the dropdown.



Start Lab Issue Log

Moderator Shriki Akarapu
Author Shikha Sharma
Reader Choose...
Inspector Choose...
Recorder Choose...

Choose...
Choose...
Shikha Sharma
Shweta Singh
Jon Brown
Andy Stefik
Eli Grant
Jim Kuznir
Patrick Ross
Samantha Curtis
Kyle Ryan
Min Kim
Zhe Dang
Shriki Akarapu

Cancel

- Both the *Author*, and the person you have assigned to be the *Recorder*, should add the issues to the paper-based review logs (which you should hand out prior to the review).

Task 6: Log issues in the Team Issue Log within an in-lab review

As team moderator, your responsibility is to lead the review of each team member's code. At the start of each review, you will lead the team in a line-by-line review of the code solution. Remember that you will strategically choose a starting point for this line-by-line

review based on (a) what sections of code have been reviewed in prior reviews (so as to minimize redundancy), and (b) what parts of the code seem particularly prone to having issues (which could be determined either by peeking at the outside-of-lab review log for each team member ahead of time, or by anticipating where students are likely to have problems.) As issues are identified by your team during the code review, have the *Recorder* add them to the paper-based Team Review Log. Here, there are two possibilities:

3. ***The team identifies an issue that an individual team member also added to the outside-of-lab review log.*** When your team identifies an issue, consult the Outside-of-Lab logs to see if the issue also appears there. If so, adding the issue to the In-Lab Team Review Log can be achieved by having the *Recorder* copy the issue from the (paper-based) Outside-of-Lab Review Log to the (paper-based) In-Lab Review Log.
4. ***The team identifies a new issue that was not identified by an individual team member outside of lab.*** In this case, have the *Recorder* add a new issue to the (paper-based) Team Issue Log.

Task 7: Transfer any remaining issues from the Outside-of-Lab Issue Log to the Team Issue Log

Be sure to stop the line-by-line review of each team member's solution at least 5 to 10 minutes prior to the end of the time allotted for the review (usually 35 minutes are allotted for each review). This will allow your team to view the Outside-of-lab Review Log of each team member, in order to see if any of the issues that individual members logged prior to the lab either were missed by the team, or simply were outside of the scope of the in-lab review. Have the team discuss each issue in a given member's Outside-of-Lab Review Log that is not present in the Team Review Log. If the team decides that the issue should appear in the Team Review Log, have the Recorder copy the issue to the (paper-based) Team Issue Log.

Task 8: Finalize the Review

Once the team completes the review of a given team member's code solution, or the allotted time is up, you need to tell OSBLE that the review is complete. Follow these steps in order to finalize a review:

Steps:

- Click on the "Finalize Review" button in the right panel. You should see the following pop-up:

Finalize Lab Issue Log

Please specify area reviewed: file names and line numbers.

- Ask the *Reader* to tell you exactly which files and line ranges (e.g., “Computations Header: Lines 1-20”) were covered in the Review. Note these in the “area reviewed” text box, one line range per line.
- If the review was videotaped, click on the “Review Videotaped?” checkbox.
- Click on the “OK” button to finalize the Review.
- Once the Issue Log has been finalized, all In-Lab Review buttons will become disabled, indicating that the In-lab Review has been completed.

CptS 121 Code Review Moderator's Guide

Version "Full" (for Wednesday Labs)
Spring, 2009 (updated 3/9/09)

Things to Bring to the Review Meeting

- Enough copies of **Code Review Checklist** for all members of your team (4-5).
- Provide students with link to assignment description available from <http://www.eecs.wsu.edu/~aofallon/cpts121/schedule.html>.
- Enough copies of the Inspection Exit Questionnaire for all members of your team (4-5).
- The attendance checklist, so that you can note which team members are present in each lab.

At the Start of Each Lab

- Read the following script to your team to remind them of the Code Review process:

Today, you will be participating in *code reviews*, a software engineering practice specifically developed to improve the quality of computer code and programming practices. In these code reviews, teams will

- (a) carefully walk through the assignment solutions of each team member,
- (b) check each line of code against a checklist of questions designed to identify potential issues, and
- (c) discuss and document the issues that arise.

Code reviews provide you with a valuable opportunity not only to improve the quality of your code, but also to improve your assignment grades. You will be required to hand in a potentially modified version of your code within seven days after the code review. Your assignment grades will be the average of the before-review and after-review versions of their code.

In any given review, I will assign you to one of the following four well-defined roles:

- **Author**—This is the person whose code solution is being reviewed. The author may be called upon to clarify his or her code as needed.
- **Reader**—This person reads the **Author's** code solution aloud, line-by-line. Readers may choose to summarize parts of the code for which they feel line-by-line review isn't necessary.
- **Inspector**—Although the entire team is charged with inspecting the **Author's** code, the Inspector takes a lead role in carefully inspecting each line of code against the provided code review checklist.
- **Recorder**—This person carefully documents the issues that arise in the review.

Here's how a code review works. The **Reader** will begin reading the **Author's** code, one line at a time. The whole team, especially the **Inspector(s)**, will check the code against the code review checklist in an attempt to spot potential issues. If you think there may be an issue, then bring it up. The **Moderator** will lead a brief (up to one minute) discussion of that issue, its severity, and its possible solutions. However, the point is not to agree on solutions; rather, it's to identify possible problems. Even if you can't agree that an issue is actually an issue, the **Recorder** must still document the issue in the issue log.

I will keep careful track of time, and stop the line-by-line portion of the review ten minutes prior to the overall time limit of the review. This will allow time for us to determine if we missed any of the issues that appear in the issue logs you compiled outside of class. We will discuss such issues, and transfer them to the Team Review log if we agree that they should be there. In addition, I will have the **Recorder** to read aloud the issues that were noted, so that the group can make sure that every issue was documented properly. I will also invite a discussion of the lessons that the team has learned from the code review, and I will have the **Recorder** note those issues.

At the end of the review, we will have compiled a team issue log that the **Author** can use to perform a revision of the code that must be submitted via OSBLE within seven days after today's lab.

Upon the conclusion of all of the team members' reviews, I will hand out an exit questionnaire that asks you to take a few minutes to reflect and comment on your experiences in the review. Once you fill out the exit questionnaire, you are free to leave.

Remember that your assignment grade will be computed as the average of the pre-review and post-review versions of the solution that you hand in, so don't forget to submit a post-review version of your solution through OSBLE.

At the Start of Each Review

- Perform introductions if participants do not all know each other.
- Distribute copies of the **Code Review Checklist** and **Assignment Description** to each team member.
- Identify the **Author** and the individuals performing the **Reader**, **Inspector**, and **Recorder** roles. There will be one **Reader**, zero to two **Inspectors**, and one **Recorder** for each review.
- Make sure that all team members are logged in to OSBLE (<http://veupl.eecs.wsu.edu/studio/>) and are viewing the author's code.
- Say:** The author has created this solution to assignment # (fill in correct number) and asked us to help make it better. Please focus your comments on improving the solution.
- Say:** Look beneath the superficial minor defects or style issues you see, to hunt out significant defects. If you aren't sure if something is a defect, point it out and we'll decide as a team.
- Say:** Our goal is to identify defects, not devise solutions. In general, I will permit about 1 minute of discussion on an issue to see if it can be resolved quickly. If not, I will ask that it be recorded and we'll move on to try to find additional defects.
- Say:** Let's have only one person speaking at a time, so there aren't multiple meetings going on simultaneously.
- Explain any attention-getting device you will use. Ask inspectors to respect your interruption and yield the floor to you.
- Say:** This review will end when all of the author's code has been reviewed or 35 minutes have passed. I will keep track of time.
- Say:** We'll take a few minutes to discuss lessons learned from the review at the end of the meeting.
- Ask everyone on the team:** If they have any questions.
- Ask the Reader:** Read the code line-by-line, starting at line x (*As moderator, it is your responsibility to pick a starting point for each review such that there is minimal overlap between reviews.*) Feel free to summarize sections of code for which you feel line-by-line inspection isn't necessary. Please keep track of exactly which lines of code you read, as we'll need to note the coverage of the review at the end.
- Say:** Anyone, and especially the **Inspector**, can jump in with a comment or question at any time. In the Code Review Log, the **Recorder** should document every issue that is raised and discussed.
- Click on the "Start Review" button to mark the time at which the review started.
- Say:** We must finish the line-by-line review by _____ (current time + 25 minutes.). Let's begin.
- Commence line-by-line review.

When the Line-by-Line Review is Done, or 10 Minutes Prior to End of Allotted Time

- Say:** Let's see if the review we just performed found all of the issues you logged individually prior to this lab.

- Say:** Please open the Outside-of-Class Review Log associated with the *Author's* solution.
- Say:** We'll go through it issue-by-issue. If we find an issue that we did not identify as a team, I will permit about 1 minute of team discussion to decide whether we should transfer it to the Team Review Log.

When Review of Outside-of-Class Log is Complete

- Say:** Please open the In-lab Review Log we just compiled. **Recorder**, please read aloud the issues, so that we can make sure the issues have been properly recorded and that we agree on them.
- Say:** Are there any lessons learned from this inspection that you'd like to note?
- Say:** I'd like the Recorder to click the "Finalize Review" button, and to note these in the "lessons learned" box.
- Say: Reader**, please tell the Recorder the names of the files and the line ranges within each file that we considered in this review. I'd like the Recorder to note these in the appropriate box.
- If the inspection was videotaped: Say: Recorder**, please check the box to indicate that this Review has been videotaped
- Say:** Recorder, please click the "OK" button to finalize the review.
- Begin the above inspection process for the next author on the team.

When All Team Members' Inspections are Complete *(Leave at Least 10 Minutes at the End)*

- Thank the team for their participation
- Administer and collect the exit questionnaire, making sure that students fill in the correct participant ID.
- Hand in exit questionnaires to Anu (the person who is doing the videotaping).

Tips for Keeping a Review on Track

- Prior to each inspection session, thoroughly familiarize yourself with the Code Inspection Checklist and particular Assignment whose solutions are being inspected. Create a list of potential pitfalls that students may run into, including specific error checks and pieces of code that students may omit. Be particularly vigilant of those pitfalls. You are the last line of defense to ensure a successful inspection! If issues, even minor ones, are glossed over, students may not find the Code Inspections as valuable.
- Rather than "schooling" team members on errors or issues, try to lead team members to discover the errors on their own by asking guiding questions such as, "I think there might be an issue here. Let's look at this code block more carefully."
- Remember that these students are early in their programming careers, and may gloss over important details. Keep Readers honest by double-checking their reading of the code, and encouraging them to delve more deeply into sections of the code that you believe could have issues.
- Encourage participation from the Inspectors, who are supposed to take the lead in identifying problems. If they don't seem to be contributing, try explicitly asking them about a section of code. For example, you might ask "Do you think that this line could have a problem?" or "Are you 100 percent confident that the section of code just read is free of problems?"
- It is OK for authors to clarify their code; however, the need to do so may indicate the presence of an issue.
- Remember that you are in charge of keeping time and pacing the inspection. A general rule of thumb is that two to five pages of code can be inspected per hour, but your results may vary.

Your Moderator script has several lines intended to remind you and your team of the time limits, but you are ultimately responsible for ensuring that a given inspection ends on time, and the next inspection begins promptly thereafter.

- If possible, have each inspection start with a different section of code, so as to minimize overlap between inspections and to maximize the total amount of code that is considered across all inspections in your group. We've found that going over the same code sections in each inspections becomes overly redundant and boring.
- You are helping us to collect data for a research data. Please remember to take attendance, double-check the review logs, note (with a "V" on the review log) the review that was videotaped, and to administer the exit questionnaire.

CptS 121 Code Review Moderator's Guide

Version "View-Only" (for Tuesday and Thursday Labs)

Spring, 2009 (updated 3/9/09)

Things to Bring to the Review Meeting

- Enough copies of **Code Review Checklist** for all members of your team (4-5).
- Provide students with link to assignment description
<http://www.eecs.wsu.edu/~aofallon/cpts121/schedule.html>.
- Two **Review Issue Logs** for each of the 3-4 inspections you will lead (10 total).
- Enough copies of the Exit Questionnaire for all members of your team (4-5).
- The attendance checklist, so that you can note which team members are present in each lab.

At the Start of Each Lab

- Read the following script to your team to remind them of the Code Review process:

Today, you will be participating in *code reviews*, a software engineering practice specifically developed to improve the quality of computer code and programming practices. In these code reviews, teams will

- (d) carefully walk through the assignment solutions of each team member,
- (e) check each line of code against a checklist of questions designed to identify potential issues, and
- (f) discuss and document the issues that arise.

Code reviews provide you with a valuable opportunity not only to improve the quality of your code, but also to improve your assignment grades. You will be required to hand in a potentially modified version of your code within seven days after the code review. Your assignment grades will be the average of the before-review and after-review versions of their code.

In any given review, I will assign you to one of the following four well-defined roles:

- **Author**—This is the person whose code solution is being reviewed. The author may be called upon to clarify his or her code as needed.
- **Reader**—This person reads the **Author's** code solution aloud, line-by-line. Readers may choose to summarize parts of the code for which they feel line-by-line review isn't necessary.
- **Inspector**—Although the entire team is charged with inspecting the **Author's** code, the Inspector takes a lead role in carefully inspecting each line of code against the provided code review checklist.
- **Recorder**—This person carefully documents the issues that arise in the review.

Here's how a code review works. The **Reader** will begin reading the **Author's** code, one line at a time. The whole team, especially the **Inspector(s)**, will check the code against the code review checklist in an attempt to spot potential issues. If you think there may be an issue, then bring it up. I will lead a brief (up to one minute) discussion of that issue, its severity, and its possible solutions. However, the point is not to agree on solutions; rather, it's to identify possible problems. Even if you can't agree that an issue is actually an issue, the **Recorder** and **Author** (who is serving as a redundant **Recorder**) must still document the issue in the issue log.

I will keep careful track of time, and stop the line-by-line portion of the review ten minutes prior to the overall time limit of the review. This will allow time for us to determine if we missed any of the issues that appear in the issue logs you compiled outside of class. We will discuss such issues, and transfer them to the Team Review log if we agree that they should be there. In addition, I will have the **Recorder** to read aloud the issues that were noted, so that the group can

make sure that every issue was documented properly. I will also invite a discussion of the lessons that the team has learned from the code review, and I will have the **Recorder** note those issues.

At the end of the review, we will have compiled a team issue log that the **Author** can use to perform a revision of the code that must be submitted via OSBLE within seven days after today's lab.

Upon the conclusion of all of all team members' reviews, I will hand out an exit questionnaire that asks you to take a few minutes to reflect and comment on your experiences in the review. Once you fill out the exit questionnaire, you are free to leave.

Remember that your assignment grade will be computed as the average of the pre-review and post-review versions of the solution that you hand in, so don't forget to submit a post-review version of your solution through OSBLE.

At the Start of Each Review

- Perform introductions if participants do not all know each other.
- Distribute copies of the **Code Review Checklist** and **Assignment Description** to each team member.
- Identify the **Author** and the individuals performing the **Reader**, **Inspector**, and **Recorder** roles. There will one **Reader**, zero to two **Inspectors**, and two **Recorders** for each review (the **Author** plus one other person).
- Make sure that all team members are logged in to OSBLE (<http://veupl.eecs.wsu.edu/studio/>) and are viewing the author's code.
- Give the **Recorder** and the **Author** a copy of a blank **Review Log**.
- Say:** The author has created this solution to assignment # (fill in correct number) and asked us to help make it better. Please focus your comments on improving the solution.
- Say:** Look beneath the superficial minor defects or style issues you see, to hunt out significant defects. If you aren't sure if something is a defect, point it out and we'll decide as a team.
- Say:** Our goal is to identify defects, not devise solutions. In general, I will permit about 1 minute of discussion on an issue to see if it can be resolved quickly. If not, I will ask that it be recorded and we'll move on to try to find additional defects.
- Say:** Let's have only one person speaking at a time, so there aren't multiple meetings going on simultaneously.
- Explain any attention-getting device you will use. Ask inspectors to respect your interruption and yield the floor to you.
- Say:** This review will end when all of the author's code has been reviewed or **35** minutes have passed. I will keep track of time.
- Say:** We'll take a few minutes to discuss lessons learned from the review at the end of the meeting.
- Ask everyone on the team:** If they have any questions.
- Ask the Reader:** Read the code line-by-line, starting at line x (*As moderator, it is your responsibility to pick a starting point for each review such that there is minimal overlap between reviews.*) Feel free to summarize sections of code for which you feel line-by-line inspection isn't necessary. Please keep track of exactly which lines of code you read, as we'll need to note the coverage of the review at the end.
- Say:** Anyone, and especially the **Inspector**, can jump in with a comment or question at any time. In the Code Inspection Log, the recorders should document every issue that is raised and discussed.
- Click on the "Start Review" button to mark the time at which the review started.
- Say:** We must finish the line-by-line review by ____ (current time + 25 minutes.). Let's begin.
- Commence line-by-line review.

When the Line-by-Line Review is Done, or 10 Minutes Prior to End of Allotted Time

- Say:** Let's see if the review we just performed found all of the issues you logged individually prior to this lab.
- I would like each of you to look through the Out-of-Lab Review Log you compiled for this *Author's* solution, and to read aloud the issues that you identified. For each issue, I will permit about 1 minute of team discussion to decide whether we should transfer it to the Team Review Log. (*Invite one team member to start, working your way around the table.*) **Recorder**, for each issue we transfer, put "Yes" in the "Transferred from Outside Lab Log" column.

When Review of Outside-of-Class Log is Complete

- Say: Recorder**, please read aloud the issues you have documented, so that we can make sure that the issues have been properly recorded and that we agree on them.
- Say:** Are there any lessons learned from this inspection that you'd like to note?
- Say:** I'd like the **Recorder** to note these in the space provided on the review log.
- Click on the "End Review" button in OSBLE.*
- Say: Reader**, please tell me the names of the files and line ranges within each file that we considered in this review, and I'll make a note of them. (*Fill in the names of the files and the line ranges.*)
- If the inspection was videotaped:** Check the "Videotaped" checkbox.
- Click the "OK" button to end the review.
- Say: Recorder**, please hand in the Review Log to me. **Author**, you may keep your copy of the Review Log.
- Say:** Everybody, please hand in to me your "Private" issue log compiled prior to lab.
- Begin the above inspection process for the next author on the team.

When All Team Members' Inspections are Complete (*Leave at Least 10 Minutes at the End*)

- Thank the team for their participation
- Administer and collect the exit questionnaire, making sure that students fill in correct participant ID.
- Group together all Team Review Logs, Private Review Logs, and exit questionnaires, and hand in to Dana (the person who is doing the videotaping).

Tips for Keeping a Review on Track

- Prior to each inspection session, thoroughly familiarize yourself with the Code Inspection Checklist and particular Assignment whose solutions are being inspected. Create a list of potential pitfalls that students may run into, including specific error checks and pieces of code that students may omit. Be particularly vigilant of those pitfalls. You are the last line of defense to ensure a successful inspection! If issues, even minor ones, are glossed over, students may not find the Code Inspections as valuable.
- Rather than "schooling" team members on errors or issues, try to lead team members to discover the errors on their own by asking guiding questions such as, "I think there might be an issue here. Let's look at this code block more carefully."
- Remember that these students are early in their programming careers, and may gloss over important details. Keep Readers honest by double-checking their reading of the code, and encouraging them to delve more deeply into sections of the code that you believe could have issues.
- Encourage participation from the Inspectors, who are supposed to take the lead in identifying problems. If they don't seem to be contributing, try explicitly asking them about a section of

code. For example, you might ask “Do you think that this line could have a problem?” or “Are you 100 percent confident that the section of code just read is free of problems?”

- ❑ It is OK for authors to clarify their code; however, the need to do so may indicate the presence of an issue.
- ❑ Remember that you are in charge of keeping time and pacing the inspection. A general rule of thumb is that two to five pages of code can be inspected per hour, but your results may vary. Your Moderator script has several lines intended to remind you and your team of the time limits, but you are ultimately responsible for ensuring that a given inspection ends on time, and the next inspection begins promptly thereafter.
- ❑ If possible, have each inspection start with a different section of code, so as to minimize overlap between inspections and to maximize the total amount of code that is considered across all inspections in your group. We’ve found that going over the same code sections in each inspections becomes overly redundant and boring.
- ❑ You are helping us to collect data for a research data. Please remember to take attendance, double-check the review logs, note the review that was videotaped, and to administer the exit questionnaire.