Guiding Principles and Pedagogical Tools for an Introductory Software Development Course

Dr. Mark Hoffman, Quinnipiac University

Mark Hoffman is a professor of computer science at Quinnipiac University. He joined the University in 2001 following a career in industry and has taught a wide variety of courses including data structures, computer architecture and organization, software development, and the senior capstone project. His research interests include communication and critical thinking skills in computer science education, and the impact of technology on work/home boundary management. He received his Ph.D. from Polytechnic University in Brooklyn, NY.

Dr. Stefan C. Christov, Quinnipiac University

Stefan Christov holds Ph.D. and M.S. degrees in Computer Science from the University of Massachusetts Amherst and a B.S. degree in Computer Science from the State University of New York, College at Brockport. He has experience in teaching undergraduate introductory computer science and engineering courses as well as upper-level software engineering courses, including software quality assurance, software project management, and software engineering in health care. His current research interests include improving the quality of human-intensive processes (HIPs), such as medical processes, with a focus on detecting human errors before harm is done and preventing such errors. He has used software engineering techniques to formally represent and analyze models of complex HIPs and industrial engineering techniques to elicit and validate models of such processes. He is also interested in human-computer interaction techniques for presenting information to assist process performers during an ongoing process. His work has resulted in publications in international journals and conferences.

Guiding Principles and Pedagogical Tools for an Introductory Software Development Course

1. Introduction

Introductory software development courses pose an interesting challenge to educators as these courses typically need to effectively combine both the technical and the social aspects of software engineering as well as introduce students to the development and the maintenance of non-trivial software systems. Organizing such courses around a team-based project is a common practice to achieve these objectives. In addition to enabling students to participate in the "practical work" of the CS and SE disciplines, they also serve a fundamental pedagogical purpose as "effective learning is based on activity" [7]. Projects provide opportunities for effective learning in multiple ways: they reinforce learning by requiring students to make abstractions concrete by applying theoretical concepts to build working systems; when successful, projects allow students to demonstrate mastery; and they also allow students to participate in professional practice [7]. Non-trivial projects allow students to solve problems associated with larger and more complex systems. In the process, students build artifacts that may be used as evidence of mastery for potential employers. For SE, in particular, CS 2013 states that "students can best learn to apply much of the material defined in the Software Engineering [Knowledge Area] by participating in a project" [6]. Several exemplars of software engineering and development courses utilize projects [6]. In addition to allowing students to develop and demonstrate technical achievement, sufficiently large projects require students to work on teams [6,7,10,19] creating opportunities to learn non-technical skills such as communication skills [12], and interpersonal, leadership, and negotiation skills [13].

The pedagogical literature on project-based computing courses primarily focuses on descriptions of specific course instances (e.g., a course offering during a single semester), on a specific project used in a course instance, or on the software development process employed within a course. There is, however, relatively little reported on high-level principles to guide the design of introductory software development courses and on effective combinations of specific pedagogical tools (e.g., assignments and assessments) to support such guiding principles.

This paper describes principles that have emerged from teaching a project-based introductory software development course over the past 12 years and also presents specific pedagogical tools designed to support these principles with the ultimate goal to promote student learning and engagement. To illustrate these tools, the paper describes several assignments and assessments from the last offering of the course. The full set of pedagogical tools is available on the web. We also report results from the last three offerings of the course based on student grades and on three kinds of student self-reported evaluations of learning achievement. These results suggest the efficacy of the overall course structure and the combination of the pedagogical tools used. We also offer lessons learned from the past 12 years of experience with the course. The main contributions of this work are the guiding principles for designing introductory software development courses, the catalog of concrete pedagogical tools that instructors can adopt for their use in similar courses, and the specific combination of these tools in an iterative process that supports student learning and seems to result in a positive student experience.

The remainder of this paper is organized as follows. Section 2 reviews related work and Section 3 describes the guiding principles that emerged from teaching the introductory software development course. The course structure is described in Section 4. Section 5 describes the pedagogical tools used to implement the guiding principles. The results of implementing the pedagogical tools and a discussion of these results are presented in Section 6. Finally, Section 7 concludes the paper and discusses future work.

2. Related Work

Project-based courses in software engineering and development have been reported in the literature. Early papers by Northrup [15] and Adams [1] describe courses where projects give students hands-on experience with programming in-the-large and with the software development life-cycle. Both use the waterfall methodology where change to the documented configuration is controlled by a control board. In both papers, the course instructor serves as project manager; however, the manager reported by Adams [1] also serves on the Configuration Control Board and the Quality Assurance Team and is heavily involved in managing the project. Students are assigned functional roles (e.g., principle architect or documentation specialist) for the project reported by Northrup [15]. Adams [1] reported that students' primary role was to review and present documents. The course project described by Northrup [15] employed a client and was followed up by an optional second course that maintained the software artifacts developed in the first course. In both cases, the course project spanned the entire course.

Brown, Wilde, and Carlin [4] describe a two-course graduate-level sequence where a software maintenance methodology is developed during the first course and implemented to maintain a software system during the second course. The course sequence was motivated by the desire to prepare students for the "real world." Students were assigned to functional maintenance teams (e.g., a software quality assurance team) that consisted of 1 to 5 students each. The maintenance process needed to be modified during the maintenance course to reduce the document and process overhead. The authors reported improvement to the maintenance process due to these modifications.

Three papers clustered in time (2002-2004) report experience implementing the Team Software Process (TSP) [10,17,18]. The motivating objectives were to gain practical experience with SE and process outcomes [17] and experience team-based software development [10]. TSP is a semester-long software development process where teams use three cycles of launch, strategy, plan, requirements, design, implementation, test, and postmortem. Each cycle's process is controlled by documentation that is reviewed and inspected. Students work on teams of 4 to 6 where team members are assigned functional roles. In one case the project uses a client [17] while in another a semi-realistic client is employed [18]. All three papers reported that instead of the recommended three iterations, they were only able to implement two in a semester. They all reported that TSP required significant process and documentation overhead.

At the time the TSP papers were reported in the literature, Reichlmayr [16] reported on the use of Agile development in a sophomore-level semester-long project. The motivating objective was to give students on-the-job experience with SE principles where teams of 5 to 6 students developed a simulated project. Reichlmayr briefly describes the postmortem process where teams use process metrics to reflect on and improve process management.

Two recent papers further report the use of Agile software development methodology for course projects [8,14]. In one paper, the authors describe an upper-level course where students use the

iterative features of Agile that allow them to repeat cycles where they "see and use tools that they can explain and check. [8]". The second paper describes a capstone course motivated by the desire for students to learn transferable skills [14]. In both cases, projects last an entire semester where students work on teams of 4 to 5. In one case students work on legacy client code [8] while in the other they work on a "quasi-real" project [14]. All three of the papers reporting use of Agile methodologies also reported that student performance improved with each iteration.

The discussions of project-based courses in the literature tend to focus on a relatively short number of course offerings (1-3) and tend to stop at describing the corresponding course. The work presented in this paper is based on observations over a longer period (12 course offerings) and in addition to describing the course, it also distills guiding principles that create a general framework for the development of project-based introductory software development courses. The concrete pedagogical tools used to implement the guiding principles are also provided whereas most of the work in the literature stops short of providing such concrete and therefore easily reusable pedagogical tools.

3. Guiding Principles

Over the 12 years that the introductory software development (ISD) course has been developed and taught, four guiding principles have emerged that have contributed to the course's present form. In this section, we explain each principle, why it is important for students' learning of software development, and how each principle emerged.

Students need to experience a wide view of software development. After completing a two semester introductory sequence of programming courses, CS1 and CS2, students often enter the ISD course with a limited understanding of or with misconceptions about software development. Students may have worked in pairs, but most of their experience is individual programming projects that solve narrowly defined problems. Features of a wide view of software development are experiencing software development as a social, rather than a solitary, activity [3], experiencing maintenance of legacy code [3,8], experiencing working on a non-trivial sized system, and developing awareness of the social, cultural, and ethical responsibilities of software developers. These features are motivated by the nature of software engineering work.

The "wide view" principle has guided the course since its first offering. We wanted students with experience limited to programming to understand that programming is a small component of the software development process. We also wanted students with solitary programming experience to understand that programming is typically a collaborative activity in the professional workplace.

Students need to learn about and through workplace practice. Novice software developers often learn by performing tasks, such as fixing bugs or implementing a non-critical feature [3]. These tasks are done within a community of practice [3,12] where novices learn by contributing to the community's work and receiving feedback from that community. For example, a novice learns to be persuasive and technically accurate when explaining to a supervisor how a bug has been fixed and arguing how it was determined that the fix was successful. Learning about and through workplace practice provides students with an understanding of the professional community they aspire to join and allows them to develop useful workplace practices that will benefit them as interns or as they transition to a professional position. The awareness that what

students are learning is applicable toward advancing their careers tends to motivate them to learn and engage in a course setting.

The "workplace practice" guiding principle was motived by the desire for students to experience software development as it is practiced in a professional workplace. Limited pedagogical strategies employed in the first iteration of the course have been replaced with more effective strategies such as workplace scenarios, described in more detail in section 5.

Students need to develop communication skills. Communication skills include the ability to read, understand, and express concepts from the technical and professional literature, the ability to clearly and accurately present technical information to a variety of audiences in written and oral form, and the ability to effectively and efficiently work in a team. Communications skills are considered a component of students' learning that will benefit them in the workplace, and one place to learn such skills is through software development projects [7,14,18]. Traditionally, software development courses have focused on writing skills (e.g., [20]), but typically little attention has been paid to how other communication skills are developed. Since communication skills are a significant and integral component of the software development workplace [3,12], they need to be developed intentionally and integrally with technical skills. Communication skills were a component of the course since the first iteration; however, they did not become a guiding principle until the integral nature of technical and communication skills became clear [12].

Students need to learn by doing, with frequent feedback and reflection. Software engineering skills tend to be highly-applied and often best learned through repeated practice [6]. Learning requires focused time on task with timely and targeted feedback that is used to improve subsequent performance [2]. As students develop and refine their understanding, occasional failures provide students an opportunity to reflect on their decisions and learn by devising and implementing improvements [2,7]. Allowing students to take ownership and make their own decisions rather than follow a highly prescribed or overly supervised plan provides them with opportunities to take ownership of their failures and improvements. Through repeated use of reflection, students learn how to learn, which is a component of the critical "lifelong learning" skill valued in the software development workplace.

The "learn by doing" component of this guiding principle was motivated by the applied nature of software development. The component related to frequent feedback and reflection is based on learning theory [2] as well as on the observation made during the initial course offerings that students, being relatively inexperienced software developers, inevitably made suboptimal decisions, but given frequent feedback and ownership of their work learned from analyzing and improving that work. This guiding principle is the main motivation behind the semester-long project used in the software development course described in the next section.

4. Description of the Introduction to Software Development Course

In this section we describe key features of the ISD course as it was structured in Fall 2016. A detailed description of the course and associated course materials are available at http://webspace.quinnipiac.edu/schristov/ISD-course.html.

4.1 Curriculum Context

The ISD course is a sophomore level course that is required for computer science majors and minors, and for software engineering majors. It is the third course in the introductory

programming sequence following Programming and Problem Solving (CS1) and Data Structures and Abstraction (CS2). For computer science majors and minors it may be their only software engineering course; for software engineering majors it is their first software engineering course. Upon successful completion of this course, students are expected to achieve the following student learning outcomes:

1) Evaluate and maintain an existing software product.
2) Develop clear, concise, and sufficiently formal life-cycle artifacts including requirements, design, implementation, and test documentation for software systems based on needs of users and stakeholders.
3) Explain the value of construction technologies such as version control and design tools to assist the software development practice.
4) Explain the purpose of testing and apply it to manage an existing software product.
5) Work on a team and communicate, orally and in writing, a software design to various audiences.
6) Explain the social and ethical implications of the software development process.
7) Demonstrate the ability to think critically and be reflective learners.

4.2 Course Project

The course project is a semester-long project that facilitates the structure of the course. The project is a maintenance project where teams receive legacy code of a software system from a team that worked on that system during the previous iteration of the course. Student teams evaluate the software system, identify bugs to fix and enhancements to add, select a development and test environment, and then spend eight weeks maintaining that software system. At the end of the project, student teams package the resulting software system to be used by a team in the next iteration of the course. Each team works on their own version of the software system. Teams consist of 4-5 students each.

4.3 Software Development Methodology

The software development methodology used in the ISD course is based on features of Scrum. The eight weeks during which teams work on the code of the software system are divided into four two-week Scrum cycles (i.e., sprints) where a working version of the software system must be demonstrated at the end of each cycle. Bugs to be fixed and enhancements to be developed during each Scrum cycle are maintained on a prioritized backlog. At the beginning of each Scrum cycle, backlog items are taken from the top and then decomposed into work units that are assigned to individual or pairs of team members. During the Scrum cycle, team members meet periodically ("daily" in Scrum terminology) for a Scrum meeting (i.e., progress report). At the end of each Scrum cycle, teams complete a postmortem where they identify significant events that helped or hindered their ability to develop the project and devise strategies to improve the process. Each team selects a Scrum master who coordinates the team's activities by performing tasks, such as facilitating Scrum meetings and monitoring work unit assignment and progress. Rather than having assigned roles beyond those of the Scrum master, team members self-organize based on responsibilities needed to complete the selected backlog items for each Scrum cycle.

4.4 Project Schedule

The course project is completed in three main phases. The first phase, weeks 1-5, consists of assignments designed to equip teams with knowledge and skills needed to perform maintenance of the software system in the second phase. During the second phase, weeks 6-13, teams

perform maintenance tasks on the software system using a Scrum-like development process. Phase three, weeks 14-15, includes packaging the modified software system for the next iteration of the course, individual reflection by team members on their participation in the course project, and final presentations and demos. A weekly course assignment schedule is shown in Table 1. Students are assigned Reading Response Questions in Phase 1 (weeks 3, 4, and 5) and Phase 2 (weeks 6, 8, 10, and 12). These are questions on assigned readings and need to be answered in writing.

Table 1. Course Project Phases and Weekly Assignment Schedule

Phase	Week(s)	Assignment	Purpose
1	1	Program Review Report	Install and evaluate the course project, analyze documentation, and identify existing bugs
	2	Customer Requirements Report	Teach prospective users how to play the game, and gather enhancement requests and bugs that need to be fixed
	3	Prioritized Bug and Enhancement Report	Synthesize results from Produce Review Report and Customer Requirements Report to develop criteria for project development. Prioritize bugs and enhancements
	4	Project Management Tools Report	Evaluate and select project management and configuration management tools
	5	Preliminary Test Plan Report	Identify key behavior of fixed bugs and implemented enhancements, and write test cases
2	6-13	Scrum Process Management	Perform maintenance tasks on the software system over 4 two-week Scrum Cycles
3	14-15	Project Final Report	Package modified software system for next iteration of course and make a final presentation.
		Course Project Reflection Report	Reflect on individual participation in course project.

4.5 Project Assignments and Assessment

The first five course project assignments are focused on technical tasks to prepare teams for the code maintenance during the subsequent Scrum cycles. (Table 1 shows the purpose of each course project assignment.) The assignments consist of a task description, deliverables, technical and communication skills learning objectives, and technical and communication skills rubrics. For each of the first five assignments teams submit a written report and give a short presentation that summarizes key results from the written report. Each assignment is distributed on Monday with a discussion of the learning objectives and rubrics. Two or three examples of graded student work from prior course iterations provides teams with performance expectations and model reports. The technical rubric for each assignment is focused on the technical task; the communication skills rubric is progressive, meaning that subsets of evaluation criteria (learning objectives) are gradually added to the rubric for each subsequent assignment until the entire communication skills rubric starts to be used during the first Scrum Cycle. For each assignment, student teams submit a written report and make a presentation of their findings. The presentation audience includes other student teams (peers) and the course instructor (project manager).

For each two-week Scrum Cycle during phase 2, the same assignment is used. Like the assignments from the first five weeks, it includes a task description, deliverables, technical and communication skills learning objectives, and technical and communication skills rubrics. At the start of the first Scrum Cycle the technical and communication skills learning objectives and rubrics are discussed with two to three examples of graded student work from prior iterations of the course. Each Scrum Cycle, teams submit written reports (initial work plan, progress reports from Scrum meetings, final work plan, updated backlog, and individual and team postmortems) and make a presentation to demonstrate the progress made on the software system and to summarize the team's work. The presentation audience includes other student teams (peers) and the course instructor (project manager).

At the end of the course project, phase 3, the final assignment focuses on the technical task of packaging the project for a team for the next iteration of the course and the communication task of presenting the team's work on the course project to an audience including invited guests not familiar with the project. The final assignment includes a task description, deliverables, technical and communication skills learning objectives, and technical and communication skills rubrics. The rubrics are discussed when the assignment is distributed; however, since this is the last assignment for the course project and teams are expected to have reached a high level of proficiency, there are no examples of graded student work available. The same communication skills rubric is used as the one being used since the first Scrum cycle.

During phase 3, students individually complete a guided course project reflection report consisting of six questions to evaluate their performance on the course project. The course project reflection report includes a grading rubric.

Reading Response Questions, assigned at various points in the semester, require students to individually read a relevant text and answer open-ended question in an online reading response journal. Responses are graded using a rubric provided for all reading response assignments.

4.6 Grading

The grading scheme is included in the course syllabus and consists of course project assignments and points allocated to each assignment. Assignments are either team-based or individual. The individual assignment are the individual postmortem, reading response questions, and course project reflection report. The remainder of the assignments are team-based where all team members receive the same grade for the assignment. Of the total points a student can receive during the semester, approximately 26% are based on individual assignments. Thus, a significant portion of a student's course grade depends on team performance.

5. Pedagogical Tools for Implementing Guiding Principles

This section describes the pedagogical tools used to implement the guiding principles described above. Each pedagogical tool is designed to promote student engagement and learning within the framework of the course project. The ranked mapping of pedagogical tools to guiding principles is shown in Table 2. The lower the number in a cell, the higher support the pedagogical tool in the row provides for the guiding principle in the column. While pedagogical tools are designed to implement one guiding principle, they could secondarily implement others.

Table 2. Ranked Pedagogical Tools to Guiding Principles Mapping

	Guiding Principles			
Pedagogical Tools	**Wider View of Software Development**	**Workplace practice**	**Communication skills**	**Learn by doing (Iterative practice with feedback)**
Maintenance Project (Student Created)	2	1	-	-
Workplace Scenarios (Integrated technical and comm. skills)	3	1	2	-
Real-world software development process (currently Scrum with postmortems)	4	1	3	2
Progressive Communication Skills Rubric (written, oral, and teaming)	-	3	1	2
Guidelines over templates	-	-	2	1
Reading Response Questions	1	-	2	-

5.1 Maintenance Project

At the beginning of the course, each team receives a partially completed software system handed off from a team in the previous iteration of the course. The course project focuses on maintaining that software system and is framed by a workplace scenario (described in more detail below). At the end of the semester, each team packages the modified software system to hand off to a team in the next iteration of the course. The packaged software system includes the code and the documentation needed to understand and maintain that system.

A maintenance project requires teams to understand sufficiently well code written by others to be able to fix bugs and implement enhancements. Students who acquire insufficiently commented or poorly structured code learn the value of comments and structure. Similarly, if supporting documentation (e.g., requirements or design specifications) is incomplete, students learn its value. New software developers are typically assigned maintenance tasks [3] making a maintenance project an appropriate introduction to workplace practice. Given that most students engaging in the project have not been confronted with non-trivial legacy code before, their view of software development is expanded beyond small, self-contained problems that will be evaluate only by the course instructor or by teaching assistants; another team will use their team's code. In this way, a maintenance project is a pedagogical tool that implements the "workplace practice" and "wider view of software development" guiding principles.

5.2 Workplace Scenarios

The course project is framed by a workplace scenario [12] where students are software developers working for a software development company. The company has acquired another software development company that has a number of partially-completed software systems. The software development project manager (course instructor) assigns teams to evaluate a number of the partially-completed systems to determine their current state and how they may be further improved. The software development project manager needs frequent reporting on each team's progress to compile reporting for upper management. A key component of the workplace scenario is the need to provide clear, concise, and accurate technical information to the software development project manager that quickly and easily satisfies the manager's reporting needs. In this way, a workplace scenario integrates technical and communications tasks. The workplace scenario and the maintenance of the handed-off software system drive the demands of the course.

A workplace scenario embeds students in a technical workplace task [12]. Workplace scenarios may be limited to a single assignment within a course or frame an entire course, as they do for the ISD course. A workplace scenario consists of five elements: *professional role for the student* (e.g., software development team member), *technical task* (fix a bug; design a test plan), *communication task* (get the manager's approval for the fix; enable the tester to conduct the test), *what the audience will use the communication for* (determine whether to approve the fix; run the test), and the *genre* the student is to use (bug report; test plan). In the scenario, the person with whom the students communicate is not the instructor but rather another employee, client, or some other person borrowed from the workplace. Table 3 shows the workplace scenario elements of the Program Review Report assignment. The Program Review Report exists within the workplace scenario that frames the course. In this case, not all partially-completed projects may be allocated resources; to decide which project will be funded the software development project manager needs an evaluation from each team.

Table 3. Workplace Scenario for Program Review Report

Professional role	Member of a software development team
Technical task	Install, learn to operate, and evaluate a partially-completed project
Communication task	Report results of evaluation to project manager
How audience will use communication	Compare the team's ability to complete its project with the abilities of the other teams to complete their projects
Genre	Evaluation Report

Workplace scenarios give students experience with "workplace practice" where technical and communication skills are significantly integrated. They also allow each student to experience working as a team member where the team's performance depends on the collective performance of all team members.

5.3 Real-World Software Development Process

Teams maintain the software systems handed off to them using features of the Scrum development process. The Scrum features selected give students and teams experience working with a modern development process [8,9]. Two features, in particular, implement the guiding principles shown in Table 2: postmortems and software system demonstrations.

Postmortems are performed at the end of each Scrum Cycle. Because time is not available for teams to meet to conduct an in-person postmortem, postmortems are managed through a process

described in the Scrum Postmortem assignment. Each team member completes an Individual Postmortem that is submitted to the Scrum Master. The Individual Postmortem consists of a timeline of three significant events (good, bad, or neutral) that occurred during the last Scrum Cycle, and an Action Plan for performance improvement. The Action Plan includes two measurable objectives: one for a practice to continue and one for a practice that needs to be revised. The Individual Postmortems are reviewed by the Scrum Master who constructs a Team Action Plan with two measurable objectives for the team.

At the end of each Scrum Cycle, teams demonstrate their current working version of the software system. In addition to demonstrating fixed bugs and implemented enhancements, team members present their contribution with a focus on code. Ten minutes are allocated to each demonstration followed by 5 minutes of immediate feedback by members of other teams and the course instructor. The evaluation is guided by a rubric that assesses preparation, technical clarity, delivery, and organization. Two open-ended questions ask what was "best" about the presentation and what "could be improved upon." Team members observing the presentation complete the rubric along with the course instructor and share their responses to the open-ended questions. The course instructor collects and reviews individual rubrics when evaluating each team's presentation performance using the communication skills rubric described in the next section.

The software development process employing selected features of Scrum give students experience with "workplace practice" found in may software development organizations. Scrum is iterative affording students the opportunity to apply feedback to improve performance. This is specifically promoted by the postmortem assignment. An understanding of continuous improvement and the ability to write measureable objectives will be a distinct advantage to interns and graduates as they transition to the software development workplace. Teams learn to persuasively demonstrate their work and understandably present code. Through all these activities students gain valuable experience with previously unfamiliar aspects of software development.

5.4 Communication Skills Rubric

Each assignment in the ISD course (listed in the Weekly Assignment Schedule (Table 1) is assessed using a rubric. For each assignment, there is a technical rubric for technical tasks specific to the technical learning objectives of that assignment, and there is also a communication skills rubric common to all assignments. This rubric is progressive and cumulative. Starting with a relatively small subset of communication skills assessed for the first assignment, additional subsets are assessed with each new assignment over the first five weeks. Newly assessed subsets are highlighted in red, those assessed in prior assignments are highlighted in green, and the remainder are black. Starting with the first Scrum Cycle, all communication skills are assessed. Using this strategy, assessment is focused on a particular subset of communication skills in the current assignment, but feedback from prior assignments is also expected to be addressed by students in the current assignment. Teams are assessed using the communication skills rubric over ten assignments. An excerpt of that rubric applied to one team's Customer Requirements Report is shown below.

COMMUNICATION ABILITIES		**EVIDENCE EXAMINED TO EVALUATE YOUR COMMUNICATIONS ABILITIES** (Note that you present your evidence in your report)				
Ability	**Details**	**Comm. Skill**	**Done Well**	**Rating and Comments**	**Needs Improvement**	**Points**
Report helps project manager make practical decisions efficiently	Provides critical information useful to the reader *Critical information is easily accessible to the reader* *Concise and appropriate writing style for project manager*	**Writing**	Contains all information useful to the reader and none that is not useful *Uses text formatting, organization (e.g., headers), and graphic devices (e.g., lists and tables) that makes information easily accessible to the reader* *Uses style and tone appropriate for the workplace*	Good formatting; however, organization of user description and instructions needs to be clearer. Appropriate style and tone.	Useful information is missing or not useful information is included *Fails to use formatting, organization, or graphics that makes information accessible to the reader* *Uses informal style and tone that is inappropriate for the workplace*	0

The "Rating and Comments" and "Points" columns are filled out by the instructor as each team is being evaluated. The rest of the rubric columns are prefilled as part of the rubric template. In this example, one communication skills learning outcome is listed in the "Ability" column. The "Details" column lists specific items used to assess the technical learning outcome. The "Done Well" and "Needs Improvement" columns list criteria for assessment. The "Rating and Comments" column contains focused feedback to the team and the "Points" column is used to report the points allocated for this item. For this assignment, "0" is the midpoint on a three-point scale meaning the team was assessed half the points for this item where "+" indicates full and "-"indicates none.

The communications skills rubric specifically implements the "communication skills" guiding principle. The rubric further gives students experience with the "workplace practice" of continuous improvement and clearly, concisely, and accurately communicating technical information.

5.5 Guidelines over Templates

For each assignment, except for the last one, examples of graded student work are available. Rather than serving as templates to fill in or mimic, these examples provide a variety of formats that may be employed by a team to fulfill an assignment. These examples also contain instructor feedback on components that worked and components that did not. When teams are provided the purpose, guidelines, and examples of various forms of a particular genre, such as a work plan, they are afforded the opportunity to "play" with the form. Focused feedback allows them to iteratively develop a form that effectively fulfills a purpose as well as an understanding of why it

works and how it was developed. Providing teams with restrictive templates forecloses on this opportunity to learn.

5.6 Reading Response Questions

Rather than drive the course and course project with a list of software development topics to "cover," the topics are integrated into and driven by the project. For example, during the third week of the course as student teams are developing criteria to organize a prioritized list of bugs and enhancements, and they are figuring out how to work as a team, they read "The Surgical Team" chapter from Brooks' *The Mythical Man-Month*. The reading introduces students to one text in the computer science canon and gives them an opportunity to think about and comment on a particular team organizational strategy. In this case, three low stakes writing-to-learn questions invite them to engage:

1. Highlight the sentence you think *best summarized* Brook's argument in the reading. Briefly (3-5 sentences) explain why you chose the sentence.
2. Highlight the sentence or phrase that you think is the *most critical characteristic* in creating a successful project team. Briefly (3-5 sentences) explain why you chose the sentence or phrase.
3. From another source, online or library, find three characteristics that your think are the *most critical characteristics* in creating a successful project team. Briefly (3-5 sentences) explain how each contributes to the success of a team.

The results are graded on a 5-point scale by the course instructor prior to the class meeting. In class, student responses may be used in a variety of ways that offer students opportunities to think critically about team organization as well as Brooks' observations.

Reading Response Questions and in-class discussion expose students to a number of software development topics unfamiliar to them: team organization and performance (e.g., Brooks' *The Mythical Man Month*), design (Norman's *The Design of Everyday Things*), and ethics (Epstein's *The Case of the Killer Robot*). The ability to read and apply concepts from a text implements one aspect of the "communication skills" guiding principle.

6. Results and Discussion

This section reports data used to evaluate in terms of student engagement and learning the effectiveness of the pedagogical tools used to implement the guiding principles. We also offer lessons learned from our experience with developing these pedagogical tools.

6.1 Results

Data was gathered from four sources: student course evaluations, student learning outcomes achievement survey, student course project reflection assignment, and average team grades for workplace scenario assignments, which are all the assignments in the course except for the reading response assignments, the individual postmortems, and the end-of-semester course project reflection report. Student course evaluation data were gathered for Fall 2015 and Fall 2016. A new student course evaluation questionnaire was implemented by the School of Engineering starting in Fall 2015 making comparison with earlier results unwise. Student learning outcome achievement data was gathered from Fall 2015 and Fall 2016. The questionnaire was implemented as part of a related study [5] starting in Fall 2015. Data was gathered for the student course project reflection assignment and average team grades for workplace scenario assignments from Fall 2014, Fall 2015, and Fall 2016. Starting in Fall 2014

the assignments and assessment process stabilized (after continual modifications since Fall 2010) and were consistently applied in subsequent course offerings, such that results can be meaningfully compared.

6.1.1 Student Course Evaluations

Student course evaluations are conducted at the end of each semester using an online evaluation form administered by the School of Engineering. These course evaluations are anonymous and course instructors do not receive results until after course grades have been submitted. Student course evaluations allow students to assess various aspects of the course. Results for relevant questions from the student course evaluations are reported in Table 4. Students answered each question on a five-point Likert scale where 5 is "Strongly Agree" and 1 is "Strongly Disagree".

Table 4. Engineering Student Course Evaluations

Evaluation Question	Fall 2015		Fall 2016		Average
	Section A (Response Rate: 14 of 16 enrolled students, 87.5%)	Section B (Response Rate: 14 of 18 enrolled students, 77.8%)	Section A (Response Rate: 16 of 17 enrolled students, 94.1%)	Section B (Response Rate: 11 of 12 enrolled students, 91.7%)	
COMMITMENT: As a student, I did my part to learn as much as possible in this course.	4.7	4.6	4.6	4.5	4.6
LEARNING: As a student I can apply information/skills learned in this course.	4.8	4.6	4.5	4.6	4.6
FEEDBACK: The professor provided timely feedback on my course work that reinforced my learning.	4.9	4.3	4.8	4.6	4.7
ASSIGNMENTS & EXAMS: The assignments and/or examinations were reflective of the course content.	4.9	4.6	4.1	4.3	4.5

6.1.2 Student Learning Outcome Achievement Survey

As a component of a collaboration between a Software Project Management course and the ISD course, students completed a survey self-assessing their achievement of the student learning outcomes (SLOs) for the ISD course [5]. The seven SLOs from the course syllabus were broken down into eight on the survey for the sake of clarity. 32 of the 34 students enrolled in the course in Fall 2015 completed the survey (94% response rate). 29 of the 29 students enrolled in the course in Fall 2016 completed the survey (100% response rate). Students answered the question "In the ISD course, I learned the following skills or knowledge" on a five-point Likert scale where 5 is "Strongly Agree" and 1 is "Strongly Disagree." Table 5 shows the results from this survey.

Table 5. Student Learning Outcomes Achievement

Student Learning Outcome	Fall 2015		Fall 2016		Average	
	Average Response	Percent "Strongly Agree" and "Agree"	Average Response	Percent "Strongly Agree" and "Agree"	Average Response	Percent "Strongly Agree" and "Agree"
1. An ability to evaluate and maintain an existing software product.	3.94	81.26	4.03	86.20	4.0	83.73
2. An ability to create software development documents including Work Plan, Backlog, and Post Mortem.	4.09	84.38	4.45	96.56	4.3	90.47
3. An ability to use tools to manage the software development process.	3.75	75.00	4.00	75.87	3.9	75.44
4. An ability to develop and apply tests to demonstrate product quality.	4.06	84.38	3.83	75.86	3.9	80.12
5. An ability to work effectively on a team.	4.00	84.38	4.38	93.11	4.2	88.75
6. An ability to communicate (written and oral) effectively in a variety of ways.	4.09	84.38	4.48	93.10	4.3	88.74
7. An ability to explain the social and ethical implication of software development.	4.13	81.26	3.79	65.52	4.0	73.39
8. An ability to think critically and reflectively.	4.09	87.50	4.21	82.76	4.2	85.13

6.1.3 Course Project Reflection Assignment

At the end of the course, students completed the Course Project Reflection Assignment. Two questions asked students to evaluate their progress on student learning outcomes:

- What student learning outcome (see the syllabus) did you make the most progress toward achieving? Provide evidence from course project documents to support your claim.
- What student learning outcome (see the syllabus) did you make the least progress toward achieving? Provide evidence from course project documents to support your claim.

The goal of these questions was to obtain additional information about achievement of the student learning outcomes that complements the information obtained via the student learning outcome achievement survey. The results for 19 of 24 students enrolled in the course in Fall 2014 and 28 of the 34 students enrolled in the course in Fall 2015 are shown in Tables 6a and 6b. The results for 28 of 29 students enrolled in the course in Fall 2016 are shown in Table 6a. The results for 27 of 29 students enrolled in the course in Fall 2016 are shown in Table 6b. All students completed the assignment; however, student responses were removed because they were unclear or unusable. Table 6c shows the student learning outcome differential: most achievement total from Table 6a minus least achievement total from Table 6b.

Table 6a. Student Learning Outcome "Most" Achievement from Course Project Reflection Assignment

Student Learning Outcome	Fall 2014 (N=19)	Fall 2015 (N=28)	Fall 2016 (N=28)	Total
1. Evaluate and maintain an existing software product.	4	4	7	15
2. Develop clear, concise, and sufficiently formal life-cycle artifacts including requirements, design, implementation, and test documentation for software systems based on needs of users and stakeholders.	2	4	2	8
3. Explain the value of construction technologies such as version control and design tools to assist the software development practice.	4	1	0	5
4. Explain the purpose of testing and apply it to manage an existing software product.	0	3	3	6
5. Work on a team and communicate, orally and in writing, a software design to various audiences.	9	10	16	35
6. Explain the social and ethical implications of the software development process.	0	3	0	3
7. Demonstrate the ability to think critically and be reflective learners.	0	3	0	3

Table 6b. Student Learning Outcome "Least" Achievement from Course Project Reflection Assignment

Student Learning Outcome	Fall 2014 (N=19)	Fall 2015 (N=28)	Fall 2016 (N=27)	Total
1. Evaluate and maintain an existing software product.	3	3	5	11
2. Develop clear, concise, and sufficiently formal life-cycle artifacts including requirements, design, implementation, and test documentation for software systems based on needs of users and stakeholders.	4	1	2	7
3. Explain the value of construction technologies such as version control and design tools to assist the software development practice.	1	8	4	13
4. Explain the purpose of testing and apply it to manage an existing software product.	8	2	4	14
5. Work on a team and communicate, orally and in writing, a software design to various audiences.	0	4	1	5
6. Explain the social and ethical implications of the software development process.	3	8	10	21
7. Demonstrate the ability to think critically and be reflective learners.	0	2	1	3

Table 6c. Student Learning Outcome Differential from Course Project Reflection Assignment

Student Learning Outcome	Most Achievement	Least Achievement	Differential (Most-Least)
1. Evaluate and maintain an existing software product.	15	11	4
2. Develop clear, concise, and sufficiently formal life-cycle artifacts including requirements, design, implementation, and test documentation for software systems based on needs of users and stakeholders.	8	7	1
3. Explain the value of construction technologies such as version control and design tools to assist the software development practice.	5	13	-8
4. Explain the purpose of testing and apply it to manage an existing software product.	6	14	-8
5. Work on a team and communicate, orally and in writing, a software design to various audiences.	35	5	30
6. Explain the social and ethical implications of the software development process.	3	21	-18
7. Demonstrate the ability to think critically and be reflective learners.	3	3	0

6.1.4 Workplace Scenario Assignment Grades

Each of the workplace scenario assignments (shown in Table 7) was submitted by a student team, each assignment was graded using the rubrics provided with the assignment, and each team member received the same grade for the assignment. The average team grades for the workplace scenario assignments are shown in Table 7. All assignment grades reported are normalized to a 100 point scale. For the first five assignments (Phase 1), students were assessed on a written report and a presentation. During the four Scrum Cycles (Phase 2), students were assessed on a work plan showing the team's planned and completed work items, updated backlog, individual and team postmortems, and working project demonstration. For the Final Project Report (Phase 3), students were assessed on a final written report, the packaged software system developed by the team, and a final team presentation. Figure 1 plots the data from Table 7.

Table 7. Average Team Grades for Workplace Scenario Assignments

Workplace Scenario Assignment	Fall 2014	Fall 2015	Fall 2016	Average
Program Review Report (100 points)	77.5	81.56	77.86	78.97
Customer Requirements Report (100 points)	83.33	90.74	83.43	85.83
Prioritized Bug and Enhancement Report (100 points)	87.5	91.00	88.14	88.88
Project Management Tools Report (100 points)	90.17	87.18	80.14	85.83
Preliminary Test Plan Report (100 points)	87.5	86.97	88.14	87.54
Scrum Process Management – Cycle 1 (150 points)	87.78	91.12	82.19	87.03
Scrum Process Management – Cycle 2 (150 points)	87.78	96.19	87.24	90.40

Scrum Process Management – Cycle 3 (150 points)	93.89	93.17	94.29	93.78
Scrum Process Management – Cycle 4 (150 points)	96.11	94.59	95.14	95.28
Project Final Report (200 points)	88.34	91.05	97.21	92.20

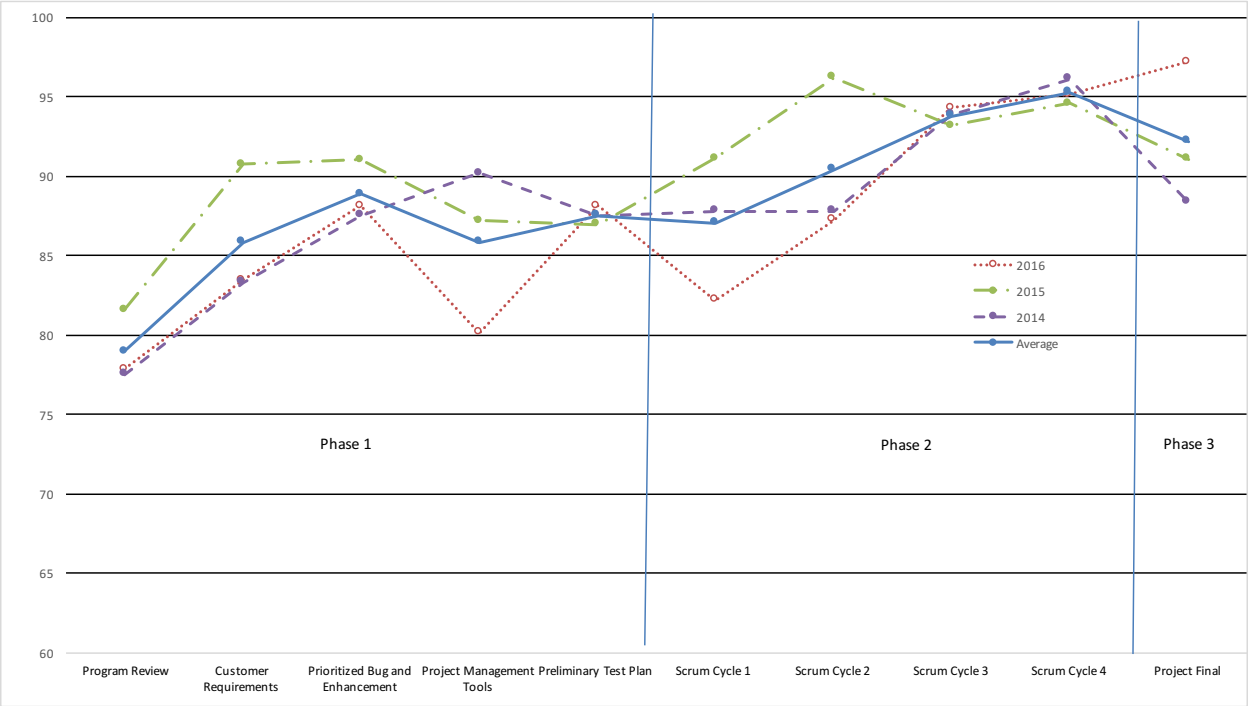

Figure 1. Workplace Scenario Assignment Grades

6.2 Discussion

This section discusses the results presented in the previous section and offers additional lessons learned based on our experience with the ISD course.

6.2.1 Overall student experience

The results in Table 4 indicate that students had a positive experience with the course. The average responses across all 4 sections are consistently high (4.5 to 4.7) suggesting that the course was a valuable learning experience.

The strong scores for the *commitment, learning,* and a*ssignments & exams* aspects support our hypothesis that the workplace scenarios providing a realistic software development experience combined with the awareness that each team "inherited" a real software system from a previous team and each team's contributions would be used by a future team motivated the students and helped them stay committed throughout the course. The strong scores for *feedback* support our assertion that frequent opportunities to iteratively practice various skills, receive feedback, and demonstrate improvement based on that feedback likely helped students maintain their commitment during the semester. These frequent opportunities for practice most likely also contributed to the high score on the *learning* aspect in Table 4, as students were given plenty of chances to apply what they learned and to recognize the improvement in their skills. We believe

that the explicit, detailed, and readily available rubrics associated with each course assignment had a major contribution toward the students' positive perception of the course grading procedures.

The scores in Table 4 for the *assignments & exams* aspect of the course are slightly lower than the scores for the other aspects listed in the table, but these scores are still high (4.1 to 4.9) indicating that students recognized the value of the workplace scenario approach to framing the course and the corresponding assignments.

6.2.2 Achievement of student learning outcomes

The results in Tables 5 and 6 provide a more fine-grained information about the students' perceived achievement of specific learning outcomes. From Table 5, students report making varying degrees of achievement toward the listed SLOs. The Course Project Reflection Assignment (Table 6) required students to select *one* SLO where they made the "most" achievement and *one* where they made "least" achievement. Even though students may have made significant progress toward all SLOs, *one* was required to be "least." Table 6c, student learning outcome differential, provides insights into student achievement ranking at the extremes (most and least achievement), but is less informative about achievement in the "middle."

Overall, Table 5 indicates that a large percentage of students believe they have achieved the learning outcomes of the course – more the 80% of students agree or strongly agree they achieved 6 of 8 learning outcomes and more than 85% of students agree or strongly agree they achieved half the learning outcomes. Table 5 shows that the top ranked SLOs are software development documents[1] (2), communication skills (5), and teamwork (6), followed by critical and reflective thinking (8), maintain an existing software product (1) and develop and apply testing (4). These results support the strong emphasis on communication skills (6 and 1), teamwork (5) and reflective thinking from feedback (8). The bottom two SLOs are management tools (3) and social and ethical implications (7). These indicate that there may be issues associate with configuration management and social and ethical implications learning.

Turning to Table 6c, student learning outcome differential, students made most progress toward achievement of communication skills and teamwork (5). Students made least progress toward achievement of social and ethical implications (6). These results are consistent with the results reported in Table 5.

Strong perceived achievement for communication skills and teamwork reported above support the combined use of workplace scenarios, a progressive communications skills rubric, a maintenance project, and a real-world software development process employed in an iterative process where students apply detailed feedback over a significant number of phased assignments. When required to identify the *one* learning outcome where they made "most" progress, excluding all others, communications skills and teamwork were most frequently selected. That means that while students reported strong achievement for the learning outcomes software development documents (2) and critical and reflective thinking (8) in Table 5, they were not the *one* that they made the "most" progress toward achievement yielding the mixed results with respect to Table 6c. While these results provide less support, they do support the effectiveness of workplace

[1] Abbreviated names of the SLOs are used in this discussion for the sake of brevity. The full names are provided in Table 5.

scenarios that integrate technical and communications tasks and feedback including post mortems.

A lower percentage of the students (75.44%) in Table 5, agreed or strongly agreed that they have achieved management tools (3) compared to other student learning outcomes. This result is reinforced by Table 6c, where a 13 of 75 respondents (over 17%) with a differential (most achievement minus least achievement) is -8 reported least progress toward achievement of the learning outcome. This suggests that software development tools, such as version control tools, need to be better integrated into the course. It might be the case that the pedagogical tool "guidelines over templates" is not effective for teaching software development tools for students at this level who also have little experience with such tools. Students in the ISD course are typically first semester sophomore students with minimal prior experience with version control or project management tools, so they may not be mature enough to successfully learn such tools on their own. More low-level guidance accompanied by some specific templates/patterns of use of such tools might be necessary.

Table 6c also indicates that a significant number of students reported making least progress toward achieving social and ethical implications (6). Two readings were selected from Epstein's *The Case of the Killer Robot*. The questions and associated in-class activities explored several ethical issues raised by the story including the ACM Code of Ethics. From the evidence in Table 6c, the time devoted to the topic was not sufficient or not effectively used signaling that this aspect of the course needs to be reevaluated.

6.2.3 Workplace Scenario Assignment Grades

Figure 1 indicates that teams tend to improve their grades over time within the different phases of the course and over the entire course. During the pre-Scrum phase (Phase 1, assignments 1-5), student grades were lower on the first assignment, but then generally improved on subsequent assignments. Similarly, during the Scrum phase (Phase 2, assignments 6-9) of the course, student grades were lower during the first Scrum cycle, but were then consistently higher during subsequent Scrum cycles. These results suggest the value of iterative practice and frequent feedback, which is part of the "learn by doing" guiding principle.

Two other trends are worthy of note. Assignments at the start of each phase are typically the lowest grades within the phase. In Phase 1, the Program Review Report is lowest, in Phase 2, the Scrum Process Management Cycle 1 Report is lowest, and in Phase 3, the Project Final Report is lowest. When students transition from the types of technical and communications tasks mastered in one phase to another there is a period of adjustment. Over subsequent assignment with each phase students apply feedback, thus improving their performance. The slight drop in average grades for the last two assignments in Phase 1 (Project Management Tools Report and Preliminary Test Plan Report) are due to the fact that students taking the course have little experience with project management tools and testing. This is consistent with the result reported by students about their progress toward student learning outcomes discussed above. These results provide support for the need for iterative practice with frequent feedback.

The results reported above support the effectiveness of the pedagogical tools used to implement the guiding principles to promote student learning and engagement. We do not, however, have direct evidence concerning the contribution of any particular tool. As is shown in the Ranked Pedagogical Tools to Guiding Principles Mapping (Table 2), individual pedagogical tools support the implementation of multiple guiding principles to varying degrees. Workplace

scenarios place students in a situation where they intentionally participate in workplace practice. The workplace scenarios embody the significant integration of technical and communication skills. Selecting Scrum features as the software development process engages students in a modern software development process while also affording opportunities to develop understanding that may be applied and refined iteratively. The key observation from the results is that the pedagogical tools work together to promote student learning and engagement.

6.2.4 Other Lessons Learned

A maintenance project has been used since the first iteration of the course. Over the 12 years that the course had evolved, there have been three projects. The first project was seeded with a small networked whiteboard module created by a departmental colleague. The project was handed off and grew into a larger course management system until Fall 2008 when it was replaced. The next project was seeded with a game (Tank Wars) from a student's independent study project. Individual teams independently developed the project providing several different partially-completed projects for subsequent teams. Tank Wars was replaced in Fall 2015 with a second student-created game project (Judgement). All teams received the same partially-completed project; however, each team developed the project based on their own criteria generating unique projects for the next iteration of the course. In Fall 2016, 4 of the 8 projects from Fall 2015 were selected to be further developed. We have found that students generally like a game-based maintenance project as many of them are familiar with and excited about playing computer games. We also found that knowing that a team is working on a real-world project developed by another team (not the instructor) and that the team's contributions will be used by a future team are major motivating factors for students. Using such projects, however, has its downside as first-semester sophomore students are often not mature programmers resulting in deterioration of the quality of the software system over time, which in turn could frustrate and demotivate subsequent teams. This observation implies that a balance needs to be struck between student autonomy in making code contributions and quality control restrictions imposed by the instructor.

6.2.5 Limitations

One limitation of our study is the setting in which it was performed. We developed the course in an environment where the number of students each year has varied from 4 to 34 (across different sections) with class section size limited to a maximum of 24 students. The results may be valid for courses offered within similar constraints; however, they may not be valid in situations where class sizes are large or where instructors or teaching assistants lack time to devote to applying the various pedagogical tools.

Sample size also limits the generalizability of the results reported in this paper even for similar class settings. 24, 34, and 29 students over 3 semesters is a relatively small sample and the population is students in computer science and software engineering programs in a small to medium sized university. While results do offer strong support among our students they may not generalize to all such students.

Another potential limitation is the course project. We were fortunate to have two students who developed two software systems, either for an independent study or for personal enrichment, and were willing to allow us to use these systems for course projects. The software system used during the most recent course offering is a partially-developed game. Care must be taken to ensure that software systems, if they are developed by a student, are sufficiently large and complex to be challenging for other students but still be manageable and motivating.

The course has been taught by one instructor over its 12 year development. This creates an opportunity for instructor bias in evaluating and reporting results. Over the past 2 years, the co-author, who does not teach the course, participated in the evaluation and reporting of results. While this does not eliminate the potential for instructor bias, it does reduce it.

There is no comparison of data on similar or alternate approaches to teaching project-based, introductory software engineering courses. The related work that most closely resembles the approach reported in this paper is Reichlmayr [16]. The approach focuses on agile software development and iterative improvement; however, there is no data collected on student learning and student experience. Other related works do gather and report student-reported data that support similar and alternate approaches; however, comparison for the purpose of evaluating approaches to an introductory course are unwise. Northrup [15] reported course- and instructor-based results such as "the course as a whole" or "instructor's effectiveness as a teacher." Data reported in Table 4 focus on student engagement and course components. Sebern [17] reports data for student-reported achievement of course learning outcomes for a two-semester upper-level course that precedes a two semester senior capstone course. Mahnic [14] reports student-reported opinion and expected achievement in a capstone course.

7. Conclusion and Future Work

This paper describes guiding principles for teaching an introductory, sophomore-level, project-based course in software development. These principles guide the course structure and are each implemented via pedagogical tools, such as assignments and assessments that promote student learning and engagement. Results from the most recent course offerings support the effectiveness of the overall course structure and the pedagogical tools used. The main contributions of this work are the guiding principles for designing introductory software development courses, the catalog of concrete pedagogical tools that instructors can adopt for their use in similar courses, and the specific combination of these tools in an iterative process that supports student learning and results in a positive student experience.

An interesting direction for future work is evaluating the effectiveness of the course structure and the pedagogical tools in larger introductory software development courses. We are also planning to continue collecting data from future course offerings of the course in our institution to increase our sample size. We are planning to address the issues with that specific course described in the *Discussion* section. In particular, the two issues raised in the *Achievement of Student Learning Outcomes* section need to be addressed. Students struggled with configuration management, and as discussed, supporting materials that provide students with adequate scaffolding need to be developed and evaluated. Students also reported least progress with the "ethics" student learning objective (SLO 6 in Table 6.) We need to evaluate the assignments and assessments to provide students with opportunities to make more progress with respect to that objective.

The Course Project Reflection Assignment contains additional questions beyond those reported in Section 6.1.3 asking students to evaluate their progress toward achieving student learning outcomes. One such question asks students: "If you could *change one thing about the course*, what would it be?" Most students offer thoughtful suggestions where many have been incorporated into later offerings of the course. We plan to continue asking students this question fully expecting that students will continue to be as thoughtful.

8. References

[1] E.J. Adams, "A Project-Intensive Software Design Course," SIGCSE '93, pp 112-116, March 1993, Indianapolis, IA.

[2] S.A. Ambrose, M.W. Bridges, M. DiPietro, M.C. Lovett, and M.K. Norman, *How Learning Works: 7 Research-Based Principles for Smart Teaching*, Jossey-Bass, 2010.

[3] Begel A. and Simon B., 2008. Novice Software Developers, All Over Again. *ICER '08*, (Sydney, Australia, September 6-7, 2008)

[4] S.M. Brown, N. Wilde, and J.D. Carlin, "A Software Maintenance Process Architecture," Proceedings of 9th Conference on Software Engineering Education, Daytona Beach, FL, 1996, pp 130-141, 1996.

[5] S. Christov and M.E. Hoffman, "Evaluating the Interaction between a Software Project Management Course and a Software Development and Maintenance Course in Terms of Student Learning and Experience," *ASEE 2016*, June 26-29, 2016, New Orleans, LA.

[6] Joint Task Force on Computing Curricula, ACM and IEEE Computer Society. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science.* ACM, 2013.

[7] S. Fincher, M. Petre, and M. Clark, *Computer Science Project Work: Principles and Pragmatics*, Springer-Verlag, London, 2001.

[8] A. Fox and D. Patterson, "Viewpoint: Crossing the Software Engineering Chasm," *Commutations of the ACM*, Vol. 55 No. 5, pp 44-49, May 2012.

[9] A. Fox and D. Patterson, "Is the New Software Engineering Curriculum Agile?," *IEEE Software*, pp 85-88, September/October 2013.

[10] T.B. Hilbrun and W.S. Humphrey, "Teaching Teamwork," *IEEE Software*, pp 72-77, September/October 2002.

[11] G. Hislop, M.J. Lutz, J.F. Naveda, W.M. McCracken, N.R. Mead, and L.A. Williams, "Integrating Agile Practices into Software Engineering Courses," [need remaining citation]

[12] M.E. Hoffman, P.V. Anderson, and M. Gustafsson, "Workplace Scenarios to Integrate Communication Skills and Content: A Case Study," SIGCSE '14, March 5-8, 2014, Atlanta, GA.

[13] R. Pope-Ruark, "We Scrum Every Day: Using Scrum Project Management Framework for Group Projects," *College Teaching*, pp 154-169, 2012.

[14] V. Mahnic, "A Capstone Course on Agile Software Development Using Scrum," *IEEE Transactions on Education*, Vol. 55 No. 1, pp 99-106, February 2012.

[15] L.M. Northrop, "Success with the Project-Intensive Model for an Undergraduate Software Engineering Course," *SIGCSE Bulletin*, 21, 1, pp 151-155, February 1989.

[16] T. Reichlmayr, "The Agile Approach in an Undergraduate Software Engineering Course Project," *33rd ASEE/IEEE Frontiers of Education Conference*, November 5-8, 2003, Boulder, CO.

[17] M.J. Sebern, "The Software Development Laboratory: Incorporating Industrial Practice in an Academic Environment," *Proc. of the 15th Conference on Software Engineering Education and Training (CSEET '02)*.

[18] N. Tadayon, "Software Engineering Based on the Team Software Process with a Real World Project," *JCSC* 19,4, pp 133-142, April 2004.

[19] J. Tan and J. Phillips, "Challenges of Real-World Projects in Team-Based Courses," *JCSC* 19,2, pp 265-277, December 2003.

[20] T.P. Way, "A Company-Based Framework for a Software Engineering Course," *SIGCSE '05*, pp 132-136, February 23-27, 2005, St Louis, MO.