Using a Class-Wide, Semester-Long Project to Teach Software Engineering Principles

Paul E. Young
Department of Computer Science
University of Central Arkansas
Conway, AR, USA
pyoung@uca.edu

Donald M. Needham
Computer Science Department
United States Naval Academy
Annapolis, MD, USA
needham@usna.edu

***Abstract*—A senior-level, project-based Software Engineering course taught at the University of Central Arkansas serves as the capstone course for the Computer Science Program and introduces students to the theory, tools, and techniques used to build large-scale software systems in a project-driven setting. Foundational to the course is the use of a class-wide, semester-long course project to emphasize the theoretical aspects of the software process and the system used for scoring student performance on the project. One project is selected for the entire class with students divided into teams of four to six students to support different functional requirement areas. A milestone-driven approach is used following a modified version of the Unified Process for project development. Student scores on the project are divided into a group score, assignable via a rubric-like grade sheet, and an individual score which is determined by the individual's effort as assigned using the task-management tool, Issue-Tracker. Experiences gained and lessons learned in teaching the course are provided as a guide for those wishing to follow a similar approach to teaching Software Engineering in the future.**

Keywords—Software Engineering; Teamwork; Course Project; Project Grading; Software Process; Software Life-Cycle; Software Engineering Pedagogy

I. INTRODUCTION

Teaching Software Engineering (SE) at its core involves providing an understanding of such traditional topics as the software life cycle, the software process, requirements and specification engineering, software design, software implementation, and software testing. Some texts, such as those by Pressman [1], Sommerville [2], and Pfleeger [3] provide a more theoretical approach in addressing these topics, whereas others such as Berzins [4] take a formal methods approach in explaining the software development process. Still others, such as Schach [5], combine the theoretical with the practical, covering these topics in the context of designing and implementing some type of course project. Having been introduced to all three methods in our study of SE, a practical approach to teaching SE was chosen, partially as a result of personal bias and partially due to the results of studies such as that conducted by Prince which indicate higher retention rates attained by active learning and collaborative approaches [6].

This paper highlights the methodology used in taking such an applied approach toward teaching SE, as used in the senior-level SE course at the University of Central Arkansas (UCA). The UCA SE course evolved from similar courses taught previously by Needham at the United States Naval Academy (USNA) and by Coppit at the College of William and Mary. UCA's course extends the efforts of Needham and Coppit to create a project-driven SE course where students learn the principles of the software development process through hands-on experience in a team environment.

The introduction of a milestone-driven process is taken from Needham's work at USNA where teaching of the principles of the software development process is inter-woven with the phases of developing a suitable software product. Needham's students would divide into a number of teams where each team could choose from among one or two pre-selected projects or suggest one of their own. From Coppit comes the idea of selecting one project for the entire class, the self-selection of the project topic by the class, plus the method used for tracking project progress and measuring and scoring the level of individual participation.

This paper explains how these methods are employed in teaching the SE course at UCA. It also describes how those efforts were expanded to define the process used for project selection, how composition of project teams was modified to enhance student leadership opportunities, and how the planned iteration of the project deliverables was used to enable students to learn from their mistakes and improve not only their sense of achievement but also the satisfaction of their customer and professor alike.

Projects are selected using a student-driven process. Students are first tasked to brainstorm ideas for the project, with each student providing a brief description of their project. From this initial list of projects, students vote to narrow the list to four or five, dependent on class size. A final project for the course is then picked from this list based on student preference and a set of criteria provided by the professor.

One of the criteria provided by the professor is the project scope. The project size must be such that it can be accomplished over the course of the remainder of the semester by a single project team comprised of all of the members in the class. Unlike some approaches where the class is divided up into small teams that either work on separate projects or where each team works on the same project separately [7], one project is selected for the entire class to work on together. Separate sub-teams may be used to develop different segments of this project's functionality, but all are responsible for the overall project success.

The project is developed using a modified Unified Process [8], dividing the project into a number of milestones with specific deliverables required for each one, as illustrated in Appendix A. In addition to the required deliverables, an oral presentation to the product customer(s) and other class members is required for each milestone. Prompt feedback is provided by the professor at each milestone, with corrections and recommendations expected to be incorporated into the next milestone's deliverables.

Adapted from Coppit's work on *Implementing Large Projects in Software Engineering Courses* [9], the UCA SE

DOI: 10.5176/2251-3043_3.3.281

course uses a modified version of the Issue-Tracker task management system for tracking project progress and for determining individual contributions to the project. The Issue-Tracker system is used to define individual tasks required to complete the project, assign them to members of the team, track progress of task completion and determine the level of work performed by each member of the team. This is used in combination with a pseudo-rubric used to gauge the quality of the work that was performed in order to assign an individual score for the project to each student.

The remainder of this paper provides a high-level description of the course plus an explanation of the outcomes expected to be satisfied by students completing it. The modified Unified Process used in developing the course project as well as an example of the milestones used to mark the various workflows or increments defining the project are reviewed. An explanation of the team-based approach used for developing the project together with the structure of the project team and mechanisms used for project participation enforcement is provided. Then, a detailing of the task management system Issue-Tracker is included to show how it is used in tracking project and individual performance and in contributing to a student's project score. Finally, a look at the lessons learned through combining and enhancing previous contributions by Needham and Coppit, as well as suggested areas for future work and improvement are provided.

II. Course Overview

A. UCA Catalog Description

The methodology described in this paper is that used in CSCI4490, a senior-level SE course at UCA. The UCA SE course evolved from SI334 taught previously by Needham at the United States Naval Academy [10] and from CSCI435 taught by Coppit [9] at the College of William and Mary. As described in the UCA catalog, CSCI4490 is a "required course for majors that introduces basic principles of software engineering, including requirement analysis, specification design, testing, and software maintenance" where a "non-trivial computer software system from initial concept to a working system is developed in a team environment." [12] Similarly, the USNA catalog offered a similar description for its SI334 Software Engineering course: "An introduction to the basic principles of software engineering. Structured, object-oriented, and formal approaches are studied, with an emphasis on life cycles, object-oriented techniques and team-oriented software development." [13] The Naval Academy also offered the IT320 course for students in the Information Technology (IT) major, with IT320 designed to only cover system analysis and design. In practice, however, IT320's syllabus closely resembled that of SI334. The Naval Academy curriculum has since been restructured with all CS and IT students required to take both IC470, which is the same course offered previously as SI334 but which is now required by both CS and IT majors, and IC480, which "is a capstone course that ties together concepts from the information technology and computer science curricula to solve a practical problem" in which "team-oriented project solutions will include the requirements gathering, analysis, design and development of a computing system involving a large, multi-layer organization using appropriate information management and computing technologies." [13]

As stated in the catalog course description, CSCI4490 is formed around teaching the basic principles of software engineering. As such, Figure 1 contains a listing of the major topics taught in CSCI4490. These same topic areas were also covered in SI334 and IT320 at the Naval Academy.

B. UCA Course Outcomes

In addition to teaching the basic principles of software engineering, the course description requires development of a "non-trivial computer software system from initial concept to a working system … developed in a team environment" as part of the catalog requirements [12]. These two major components of the course are satisfied using an integrated active learning approach where students learn the fundamentals of software engineering practice while putting them to use in building a large-scale project involving all members of the class. This integrated active-learning approach is used to satisfy the expected course outcomes contained in Figure 2.

CSCI4490 includes the following major topics:

- Introduction, Scope of Software Engineering
- Software Life Cycle, including classical and contemporary life-cycle models
- The Software Process, including the classical and object-oriented paradigms
- Project Management, including the project management life-cycle, measures of project success, project management tools and techniques, configuration management, and version control
- Requirements, to include techniques for soliciting customer needs, the use of rapid prototyping to validate customer requirements, and the employment of use-cases for capturing customer requirements
- Classical Analysis, including informal, semiformal and formal methods for capturing a system's specification
- Object-Oriented Analysis, including functional, class and dynamic modeling
- Classical Design, including architectural design, detailed design and design testing
- Object-Oriented Design, covering interaction diagram development, Program Description Language (PDL) creation and detailed design testing
- Effective Module Design, outlining the evolution in the development from modules with high cohesion and low coupling to the creation of highly reusable objects
- Non-Execution Based Testing to include walkthroughs and code inspections
- Implementation, including the choice of appropriate programming language, the use of good programming practice, and alternative integration strategies
- Execution Based Testing, including glass-box and black-box testing and the use of equivalence classes and boundary value analysis to optimize test case selection
- Program Correctness Proofs, to include loop invariant determination

Figure 1. CSCI4490 Software Engineering Course Syllabus

Upon successful completion of this course, students shall be able to:

- Demonstrate an understanding of the basic components of the software life cycle and how to document each step of the software process.
- Demonstrate an understanding of the techniques for planning and monitoring the progress of a large-scale software project.
- Demonstrate an ability to work as a team and to focus on getting a working project done on time with each student being held accountable for their part of the project.
- Demonstrate an understanding of the ethics and societal impact involved in writing software in a team environment, integrating code, and giving appropriate credit.
- Demonstrate an ability to communicate their ideas effectively both with members of the development team and with actual or potential customers from both inside and outside the computing profession.
- Demonstrate a self-directed ability to acquire new knowledge in computing, including the ability to learn about new ideas and advances, techniques, tools, and languages, and to use them effectively; and to be motivated to engage in life-long learning.
- Demonstrate an ability to conduct execution and non-execution based testing, including the use of program correctness proofs, to assess the extent to which the product under development meets its requirements and specification."

Figure 2. CSCI4490 Software Engineering Course Outcomes

III. COURSE PROJECT

The foundation used for teaching the software engineering principles contained in Figures 1 and 2 is the course project. The course project provides the platform for satisfying the course requirements outlined in the catalog description and further defined in the required outcomes for the course. This section describes the details of how the course project has been employed in CSCI4490 and how students are graded on its implementation.

The software process describes the way a particular software product is built. Thus, the implementation of the course project for CSCI4490 is first described in terms of the software process. Included with the description of the course project is the mechanism used to grade students' work on the project.

A. The Course Project Software Process

As stated by Schach, the software process includes the methodology used for product development, the underlying software life-cycle model, the people used in its development, and the techniques and tools used in building the software [5]. Each of the components of the software process used in developing the course project is discussed next.

1) Methodology. The methodology used for software development is often used to describe a component of the software process and has been used to distinguish such practices as modular or procedural styles from functional or object-oriented practices. As mentioned previously, a modified version of the Unified Process [8] is used in developing the course project. The Unified Process, originally introduced as the Unified Software Development Process (USDP) by Jacobsen, Booch, and Rumbaugh [8], is the de facto standard for building object-oriented software products. It defines a two-dimensional life-cycle model for the development of object-oriented systems, loosely corresponding to the iteration and incrementation life-cycle model [5]. The first dimension defines five workflows which are used to describe the type of activity to be performed in each workflow, such as requirements, analysis, design, implementation, and test. The second dimension defines the timing of the activities to be performed. In practice four phases are typically defined, corresponding to the increments in a standard incremental and iterative life-cycle model [5].

2) Model Used for CSCI4490 Project Development. The model used for development of the course project included several components. First, a milestone-based approach was used to set a specific timeline for the completion of the project's components. Next, size of the projects was specifically chosen to best simulate the situation students might be exposed to in a corporate environment. Finally, the methodology used for selecting the course project was purposely selected to maximize the "buy-in" of students to the project on which they would be working.

Milestone-Based Approach: *In CSCI4490, four phases are defined for the class project- inception, elaboration,* construction, and transition. A milestone-driven approach is used to mark the completion of the phases as indicated in Figure 3. The methodology used is referred to as being a modified version of the Unified Process for the following reason. As depicted in Figure 3, the end of each of the phases mentioned above is not clearly defined by a milestone. Generally, the theory, techniques, and tools needed for completion of the project are provided on a just-in-time basis. This, plus difficulty in applying the object-oriented paradigm to a real-world problem causes many students to struggle to complete a workable set of UML project diagrams [23] on their first attempt. However, the ability to provide a usable set of these diagrams to the

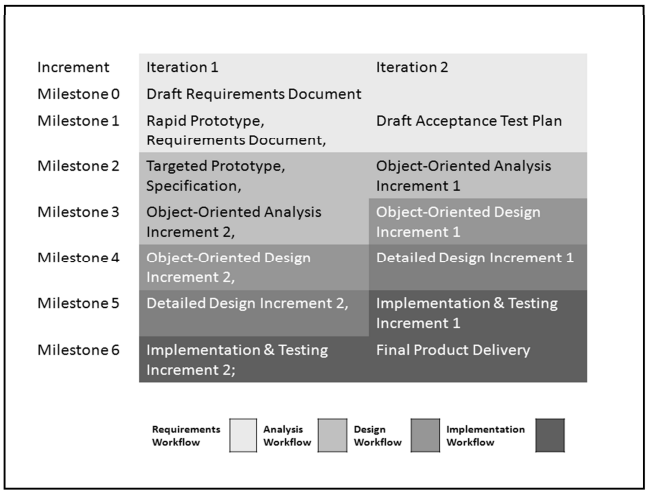


Figure 3. CSCI4490 Course Milestones

detailed design team is critical. Consequently, Analysis, Design, and Implementation are each split across two milestones in order to enable students to learn from their mistakes and improve the likelihood of producing an acceptable product at the end. Thus, the milestones incorporate planned iteration into a basic incremental approach where the milestone-based increments are taken from the four phases of a typical Unified Process project and the planned iterations allow for rework of the most critical project deliverables. The type of work performed in each milestone will likely span multiple workflows, as seen with the Analysis and Design Workflows in Milestone 3 and the Design and Implementation Workflows in Milestone 5.

a) Small vs. Large Project Approach: As pointed out by Coppit, there are two competing schools of thought in assigning capstone projects, which he refers to as the "small project" vs. "large project" approach to project selection [9]. While the small project approach is generally considered to be more manageable, both from the student's and the professor's perspectives, choosing a project that can be completed by a small group of four to six students over the course of the semester has several disadvantages over one which requires a significantly larger number of students to complete. First and foremost is the lack of communication experience gained in the small project setting. A common comment received from industry managers concerns the lack of teamwork experience exhibited by many graduating students [14][15]. Second, as members of the small project team generally understand most, if not all, aspects of the project, they can fail to see the importance of providing quality up-front deliverables that another team will be able to understand. Finally, because members of the small project team understand most aspects of the project, they fail to gain experience in how to handle problems that arise when no one individual is able to understand the entire project [9].

The course inherited by Young at the Naval Academy used the "small-product" approach. However, upon hearing Coppit's presentation at SIGCSE '05 [11] and relating it to prior experience working with industry in developing large-scale software systems, the importance of the "large project" approach was underscored and subsequently adopted in teaching CSCI4490 at UCA. Project selection and thus team composition chronicled in this paper utilizes the "large project" approach, with a single team containing all of the members in the class used to work on the same project.

b) Project Selection: In keeping with the decision to follow the "large project" approach in CSCI4490, a decision then had to be made as to how the project was to be selected for each course. Should the project be selected by the professor, or should students select what they would work on over the course of the semester? Following the model suggested by Coppit [9], project selection for CSCI4490 has been done using a student-driven process.

The project selection process begins as early as the semester before students take CSCI4490. For the last homework assignment of the pre-requisite "Object-Oriented Software Development with Java" course, students are asked to provide a brief description of a project they would like to work on in Software Engineering. The primary purpose behind this assignment is to have students begin the thought process of project selection early in order that they might begin the semester with some project ideas in mind, rather than starting with a blank slate.

The project selection process then continues in earnest on the first day of the semester with the first homework assignment and with Milestone 0. For the homework assignment, due the next time the class meets, students are required to submit a one to two paragraph description of a project that they think would be suitable for the class to consider for the course project. They are permitted to either resubmit the project idea they provided at the end of the previous semester, or to submit a new or different idea if desired. For Milestone 0 students are required to expand the description of the project idea provided in the first homework assignment to include at least five functional requirements the proposed project should satisfy. Milestone 0 is ideally due one week after the start of class in order to maximize the time available to later, more involved milestones. During the Milestone 0 review, three to five candidate projects are selected to continue into Milestone 1. The number of candidate projects allowed to advance to Milestone 1 is dependent on class size, with typically four to six members chosen to work on each Milestone 1 candidate.

For Milestone 1, and this milestone only, teams work separately from each other, with each team producing an independently developed draft Requirements Document, as specified in Appendix B, as the deliverable for this milestone. Team membership may be based on a class member's interest; each person can select the team whose project idea best interests them on a first-come, first-served basis, with a balancing of team numbers guided by the professor if needed. For the remaining milestones team membership shall be chosen by the professor in consult with the project manager. As a means of clarifying and defining the requirements presented in the Draft Requirements Documents delivered as part of this Milestone, teams develop and demonstrate a Rapid Prototype of the functionality presented by their proposed project.

Following this milestone review, one of the candidate projects being presented is selected as the course project for the remainder of the course. Selection is made based on student input, quality and completeness of the Requirements Document and Rapid Prototype, and suitability for use as a course project (size, complexity, ability to implement in Java, Java Swing graphics, web interface, etc.). Products which benefit the university, and can identify a willing product customer receive special consideration.

3) Team Structure: As discussed by Schach, the people used in the development of a software product are a key component of the software process and are critical to its success [5]. This is also true in CSCI4490 and a large degree of thought was spent in determining the project approach and subsequently the structure to be used by members of the class on the project. Following selection of the course project, the team structure is defined for the class. Following the structure first recommended by Coppit [9], the organizational structure used by the class for the team project consists of the following components:

- Customer- Preference shall be given to those projects that have an identifiable customer from outside of the class to interact with and pose questions that an informed consumer might raise. The customer would normally be expected to attend all project milestone deliveries dealing with requirements issues or user interface design. In the absence of an identifiable outside customer or other willing faculty member, the professor serves as customer.

- Program Manager- The professor serves as the head Program Manager, providing guidance to the Project Manager and Team Leaders in executing the project's methodology and resolving disputes and issues that cannot be resolved by the Project Manager.
- Project Manager- The Project Manager shall be responsible for the overall conduct of the class project. The Project Manager is selected by the professor and is expected to serve for a period to be determined by the needs of the project, generally for a minimum of two milestones. When possible, the Project Manager is a graduate student taking the graduate companion to CSCI4490, preferably one who has previously taken CSCI4490 as an undergraduate.
- Team Leaders- Team Leaders are responsible for the conduct of their team in executing a functional subset of the overall project requirements as directed by the Project Manager. Team leaders are responsible for defining and analyzing the project's requirements and for specifying a design for the product. Team Leaders are also responsible for the successful integration of their component into the overall execution of the project. Team Leaders may be chosen from available graduate students taking the graduate companion to CSCI4490, or rotated among undergraduates otherwise functioning as Developers. If rotated among undergraduate Developers, they should expect to serve a maximum of two milestones as Team Leader, project needs permitting.
- Developers- Developers shall be responsible for carrying out the directions provided by their Team Leaders in implementing the design for the project and testing the result. The typical role for a CSCI4490 student shall be as Developer.

a) Selecting Team Members: Team leadership and members shall be selected by the professor based on a "quality-spread" approach. In this approach class members are evaluated based on performance in previous courses. At UCA this evaluation is made easy since the CSCI4490 professor also teaches the pre-requisite Java course. However, in other situations where this is not the case, such an evaluation could be conducted by comparing GPAs earned on courses in computer science, using recommendations by other professors, etc. In the "quality-spread" approach team composition is chosen such that no one team is populated with only top or marginal performers; instead teams are chosen such that the "quality" of the performers is spread evenly among the teams such that each team has a relatively equal chance at success.

b) Regulations to Eject Non-functioning Team Members: While the "quality-spread" approach is designed to prevent any team from having an advantage over another, it can potentially have the unintended consequence of enabling a lower performing student to "hide" beneath the cover of the work performed by their stronger performing teammates. In an effort to curb such occurances, Needham introduced regulations to eject non-functioning members from a team, thereby requiring them to complete the project in its entirety on their own. Communication of violations to the regulations consisted of the following:

i. a Warning Memo to notify a team member that the majority of his/her team felt that they were failing to meet their responsibilities as a team member;

ii. an Ejection Memo notifying a non-performing team member that they were being removed from the team and thus required to complete the project on their own; and

iii. a Relapse Memo notifying a team member who had corrected their ways following receipt of a Warning Memo but subsequently fallen back into non-conformance.

The Relapse Memo would then be followed by an Ejection Memo removing the individual from the team. Each of these memos was to be written by the Team Leader, with copies to the Project Manager and professor, based on a majority vote of the team. [10]

4) Techniques and tools used in building the software: In addition to teaching students about the principles of software engineering and the techniques and challenges of team-based work, CSCI4490 also attempts to introduce students to the techniques and tools used to build large-scale software systems. The course provides experience in working with currenttly used operating environments, development environments, UML authoring tools, version control software, project management software, and task management systems.

a) Operating environments: One of the primary decisions for the curriculum was to use the Java 2 Enterprise Edition (J2EE) for project development [16]. This decision was made for a number of reasons. First, Java is used by a diverse number of companies from Amazon [17] to Google [18] to Yahoo [19]. Second, for Graphical User Interface (GUI) based applications (which the majority of projects selected to date are), Java offers a rich library for constructing GUI components with an accompanying event handling mechanism to enable dynamic interaction between components. Third, using J2EE introduces students to advanced concepts such as web services, database interconnectivity, and remote process invocation which they did not see in their introductory Java course, but which are important technologies in the workforce. Finally, as part of the requirement to teach a second language in the UCA curriculum, using Java in CSCI4490 reinforces the use of the concepts taught in the introductory Java course.

b) Integrated development environment: Although independent text editors, compilers, linkers, debuggers, etc. still exist and are often preferred by a large segment of software developers, software development today has evolved from the era of independent software development tools into a world where these tools are captured in an integrated development environment (IDE). For the implementation and test workflows, instruction was provided using Sun's Netbeans [20] IDE. Students were also permitted to use other IDE's such Eclipse [21] and JBuilder [22] if desired, although it was highly recommended that a single IDE be selected by the class for building the project.

c) UML authoring tools: A cornerstone in the use of the Unified Process for software development is the Unified Modeling Language (UML). UML is the de facto standard

for specifying, constructing, visualizing, and documenting object-oriented systems. At the heart of UML is a series of diagrams, from Use Case diagrams used to capture a system's requirements to Class diagrams used to specify the structure used in modeling the entities of the system [23]. As a graphical language, assistance in constructing, modifying and retaining the various UML diagrams used in system development is critical, especially as systems grow beyond the "toy" systems constructed in most Computer Science class projects. As an introduction to such UML authoring tools, students in CSCI4490 are provided with a license to use Visual Paradigm™, a full-featured UML authoring tool, although other tools such as ArgoUML [24], Violet [25], and Microsoft Visio™ [26] are also available for use. Again, as with the IDE, it is strongly suggested that students standardize on a common tool for the class, preferably one which provides for the sharing of documents across the class.

d) Version control software: One of the key components of a project involving multiple developers is the use of some sort of version and revision control software. Such software enables multiple users to work on the same program or documentation without fear of having recently submitted updates being deleted or overwritten by someone else working on the same document. Version control principles following the client-server model are taught in CSCI4490 using Subversion [27] and TortoiseSVN [28]. Adding instruction on the distributed approach to version control is planned for future courses using Git [29] or similar applications.

e) Project management software: Software to help control when tasks are scheduled, how resources are allocated, how budgets are controlled, and to support decision making during project execution, plays a vital role in ensuring projects are delivered on-time, within budget, and that meet specifications. The use of such tools as the work breakdown structure (WBS), and Gantt and PERT charts are critical in meeting such expectations. Project Management software such as is available with Microsoft Project™ [30] provides the capability to perform such tasks. Microsoft Project™ has been the application of choice for CSCI4490 up to now. Although it is proprietary software, Microsoft Project™ has been readily available under the MSDN Academic Alliance to which the UCA Computer Science department subscribes.

f) Task management software: Project size and complexity are two of the principle factors used in determining which project is selected at Milestone 1 for the remainder of the project. One of the premises by which a project can be chosen whose scope is larger than that which is typically seen for a course project, is that each member of the project team would contribute their "fair share" of effort in developing the project. One tool available for use in enforcing this assumption is the aforementioned Regulations to Eject Non-functioning Team Members. Prior to requiring the use of such an enforcement tool, the individual performance of team members can be tracked using a modified version of the task management system Issue-Tracker and thus discourage such "tailgating" practices from occurring.

i. Issue-Tracker: Issue-Tracker is a modified web-based task management system borrowed from Coppit [11]. The original system allows a project's tasks to be assigned to different members of the project team, with task progress tracked as they are completed by the developer, checked by their team leader, and approved by the project manager. The original system also provided the capability to attach supporting documentation to a task to assist in its completion or to provide to project leadership as evidence of task completion. The primary modification to Issue-Tracker provided by Coppit was the addition of a point system by which a task's value could be set based on its difficulty and importance to the project. This value could then be used to compare the work completed by the various members of the team to ensure tasks were being assigned equitably or if not to make appropriate adjustments to an individual's project score.

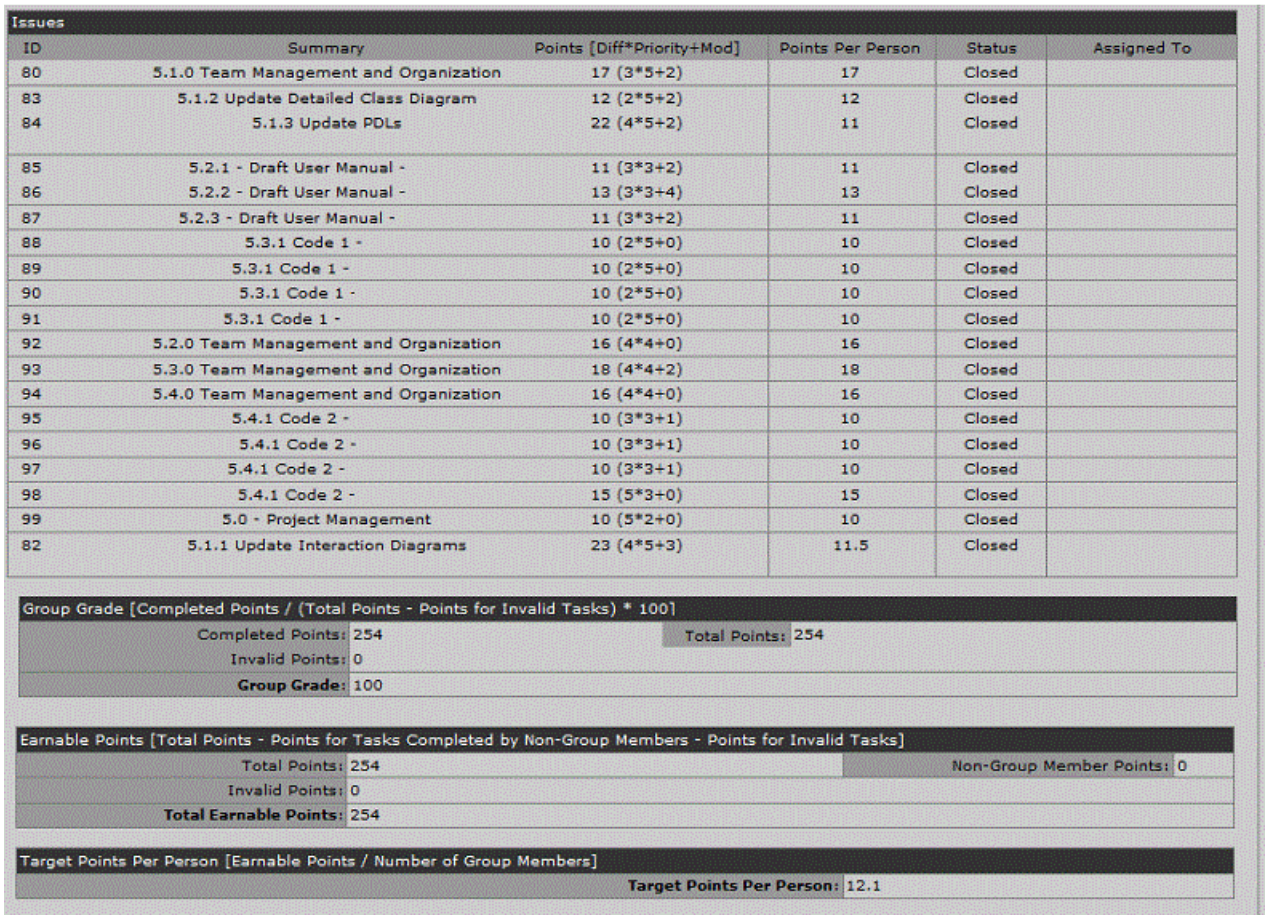
When a task is created, it is assigned a point value based on its difficulty and importance to the project. The difficulty value is simply set based on an estimate of the number of person-hours that would be required to complete the task; although conceivably some other rubric-based scheme could be used to set this value. The importance value is set based on the level of urgency required for completing this task during the current milestone. An importance value of between one and five is generally assigned to the task, with higher priority tasks being assigned a higher value for importance. Keep in mind that a task which may be considered low priority at the beginning of the development, such as developing the User Guide, may be deemed of the highest priority at Milestone 6.

The score for a task is calculated by first computing the product of the difficulty and the importance values. Then a modifier, used for rewarding exceptional performance or penalizing sub-par effort, is added to this product to determine the final score for the task. An individual's "fair share" of the work can then be calculated by adding up the points for all tasks required to complete the project, dividing those by the number of people working on the project, and comparing this average to the points earned by an individual. This "fair share" value can also be calculated at any milestone as well as a running total be maintained so that a team member may gauge his/her performance at any point during the development. [11] Figure 4 provides a sample snapshot of the points earned on their project's Milestone 5 tasks by students in UCA's Fall 2013 CSCI4490 class.

ii. Comparing Issue-Tracker to other individual performance measurement methods: How does Issue-Tracker compare to other methods for measuring individual performance on group projects? Hayes, Lethbridge, and Port defined seven criteria for evaluating individual performance on group projects and compared eight common methods used by instructors against the criteria [31]. Of the eight methods, "Quiz in class" scored highest with "Each evaluates self and others" tied for second with three other methods.

Smith and Smarkusky introduced their "Competency Matrices for Peer Assessment" which is a variant on the "Each evaluates self and others" method identified by Hayes, Lethbridge, and Port. They compared their Competency Matrix approach to the traditional peer assessment model in a student survey which showed that students favored the Competency Matrix method [32].

In Yip, Young, and Marupally [33], a subjective evaluation compared Issue-Tracker with the Competency Matrices for Peer Assessment approach. In the assessment,



Issues

ID	Summary	Points [Diff*Priority+Mod]	Points Per Person	Status	Assigned To
80	5.1.0 Team Management and Organization	17 (3*5+2)	17	Closed	
83	5.1.2 Update Detailed Class Diagram	12 (2*5+2)	12	Closed	
84	5.1.3 Update PDLs	22 (4*5+2)	11	Closed	
85	5.2.1 - Draft User Manual -	11 (3*3+2)	11	Closed	
86	5.2.2 - Draft User Manual -	13 (3*3+4)	13	Closed	
87	5.2.3 - Draft User Manual -	11 (3*3+2)	11	Closed	
88	5.3.1 Code 1 -	10 (2*5+0)	10	Closed	
89	5.3.1 Code 1 -	10 (2*5+0)	10	Closed	
90	5.3.1 Code 1 -	10 (2*5+0)	10	Closed	
91	5.3.1 Code 1 -	10 (2*5+0)	10	Closed	
92	5.2.0 Team Management and Organization	16 (4*4+0)	16	Closed	
93	5.3.0 Team Management and Organization	18 (4*4+2)	18	Closed	
94	5.4.0 Team Management and Organization	16 (4*4+0)	16	Closed	
95	5.4.1 Code 2 -	10 (3*3+1)	10	Closed	
96	5.4.1 Code 2 -	10 (3*3+1)	10	Closed	
97	5.4.1 Code 2 -	10 (3*3+1)	10	Closed	
98	5.4.1 Code 2 -	15 (5*3+0)	15	Closed	
99	5.0 - Project Management	10 (5*2+0)	10	Closed	
82	5.1.1 Update Interaction Diagrams	23 (4*5+3)	11.5	Closed	

Group Grade [Completed Points / (Total Points - Points for Invalid Tasks) * 100]

Completed Points:	254	Total Points:	254
Invalid Points:	0		
Group Grade:	100		

Earnable Points [Total Points - Points for Tasks Completed by Non-Group Members - Points for Invalid Tasks]

Total Points:	254	Non-Group Member Points:	0
Invalid Points:	0		
Total Earnable Points:	254		

Target Points Per Person [Earnable Points / Number of Group Members]

Target Points Per Person:	12.1

Figure 4. Sample Issue-Tracker Points Earned for Milestone Tasks

six of the seven criteria identified by Hayes, Lethbridge, and Port [31] were used to measure the suitability of a particular method for use in grading an individual's performance. Criteria included fairness of the method, consistency of results, level of feedback provided to the student, level of encouragement provided to students for performance improvement, resistance to grade inflation, and the grading overhead required to use the method. In the comparison, Issue-Tracker was rated as satisfying five out of six criteria as opposed to Competency Matrices satisfying only two [33].

Yip, Young, and Marupally's subjective evaluation of the two approaches was followed by a statistical comparison of the relationships between scores obtained using the two methods over the timeframe of two semesters to determine whether they provided similar results. In this evaluation, scores were compared using Pearson's and Spearman's correlation coefficients [34] to determine the degree of correlation between them. Results of this comparison indicated that the two methods did, on average, generally correspond although they did not provide the same values for an individual in all cases [35].

B. Grading Work

The Issue-Tracker point system used to determine a student's "fair share" of the effort can also be used in determining a student's score on the course project. In CSCI4490, a student's score on the project is computed by using a combination of Class Grade, which is the same for everyone in the class, and an Individual Grade computed using the points they earn in Issue-Tracker from the tasks they have completed. The final project score is the average of the Class Grade and the Individual Grade.

1) Class grade. For each milestone, a Milestone Grade is assigned using a modified rubric called a Milestone Grade sheet. For each milestone, a Milestone Specification Document is provided for the next milestone at the end of the milestone presentation. In the Milestone Specification Document, an example of which is provided in Appendix C, the list of deliverables for the next milestone is provided together with guidance to be used in completion of the deliverable. For each Milestone Specification Document, a corresponding Milestone Grade sheet is defined establishing point values to be assigned for various deliverable components. This grade sheet serves as a sort of modified rubric, although the various performance levels for each area assessed are not explicitly stated due in large part to the varying point scale used for each deliverable component. From this grade sheet, a Class Grade is computed for each milestone which shall be the same for all members of the class. The average Class Grade for all milestones at the end of the course is combined with a student's Individual Grade to determine their score for the course project. Appendix D contains a sample Milestone Grade sheet for the above referenced Milestone 2 Specification Document

2) Individual grade. To determine the student's individual grade, ,a total of all of the points earned in Issue-Tracker for each of the students in the course is first

calculated, as seen in Figure 4. Then, this total is divided by the number of students in the course to determine the target points per person. For each student, the points earned by the student are divided by the target points per person to provide a percentage score for that person. This value is the Individual Grade for that student which is then averaged with the Group Grade as discussed above to provide an overall project score for that individual. Intermediate scores can be determined at each milestone using the same technique in order that the individual may be kept aware of their standing in the class at all times.

IV. Project Successes and Lessons Learned

In the process of developing this Software Engineering course over the timeframe of the last eight years, a number of successes have been observed as well as several lessons learned. During this time different approaches have been tried, particularly with regards to the course project, with varying levels of success. Teaching the theoretical concepts of software engineering while actively employing them in the context of a semester-long course project definitely fits among the success category.

The experience of working as a team on a single project involving all members of the class, also ranks as one of the greatest achievements of the course, eliciting praise from current students regarding how much they had learned in the course and from former students on how the course had prepared them for subsequent employment.

Use of the task management system Issue-Tracker, while considered a success from the professor's perspective, did not always invoke similar positive responses from students. While Issue-Tracker helped the professors solve the problem of how to measure individual performance on group projects and scored comparably with other approaches for resolving this perpetual problem, students initially are confused on how the system works and periodically complain about the additional workload required to identify, track, and resolve issues using the system.

Issue-Tracker, in conjunction with regulations for removing non-performing team members, also helped unmask those attempting to hide under the efforts of stronger performing team members, although they did not completely resolve the problem. Finally, using members of the class to fill Project Manager and Team Leader positions on a rotating basis provided invaluable leadership and management experience to those members, enhancing their abilities for future leadership roles as well as helping them to better understand and appreciate those serving in those roles.

Along with the elements listed above as successes, there were lessons learned which were used to improve the operation of the course or can be used to further improve its operation in the future. First among these relates to the process used to fill Project Manager and Team Leader positions. While it was initially believed that it would be best to take advantage of the additional experience offered by graduate students to fill these positions, this did not always prove to be the case. First, because many of the early graduate students had undergraduate software engineering courses that were not project-based, they lacked experience performing in a team environment from which they could draw upon in leading the course. In fact, some of the better undergraduates demonstrated stronger leadership skills than those that had been placed in positions above them. This sometimes created friction between the developers and leaders. Also, by not rotating the leadership positions among the class members, most of the undergraduates were denied the valuable experience that comes from being responsible for the performance of a group. This approach might have been more successful had the graduate students been previously given experience on a team-based development project such as offered by CSCI4490.

In early offerings of the course, students struggled to complete a workable set of UML project diagrams from which to implement their projects. Upon observing this, it was decided to split the analysis, design, and implementation workflows across two milestones, addressing half of the requirements for each workflow in the first milestone and half in the second. This enabled students to learn from their mistakes in the second milestone and resulted in a better chance of delivering a product that met project requirements.

Another lesson learned is the specification of the regulations used to motivate non-functioning team members. While in principle, the regulations seem to be a beneficial tool for getting the attention of someone who is not handling their share of the responsibility in completing the project, there are two issues with the way the regulations were originally written First lies in the specification of the issuing authority for "warning" and "ejection" memos. Although used on at least two instances at UCA, there were probably more occasions where a "warning" memo should have been used but was not due to reluctance of a Team Leader to "fire" a classmate. This could potentially be resolved by having the professor serve as the issuing authority for the memo with the Team Leader or Project Manager notifying the professor of any instance of sub-par performance. Second is the need for a relapse memo. As worded, the "relapse" memo is followed immediately by an "ejection" memo. Therefore, the two memos should be combined, with relapse included as a condition for ejection.

A further lesson learned was the initial method used for Project Manager and Team Leader grading. Initially, the Individual Grades for those assigned to these positions were determined using a seven category score sheet, seen in Appendix E. These proved problematic on two accounts. First, scores were assigned to these categories subjectively, so providing substantial evidence to justify scores in each of these categories often proved difficult. Second, because categories "Adherence to project schedule" and "Quality of project deliverables" were generally related to the common overall project performance, it was sometimes difficult to differentiate between team leader performance, even if such differentiation were warranted. Finally, when combined with the overall course grading requirement for each milestone, the time required to do a detailed evaluation of the team leadership made it sometimes difficult to provide feedback in a timely manner, particularly towards the end of the project. As a result, the method for providing Individual Grades for these positions was changed so that the team leadership was required to identify those tasks they performed and enter them into Issue-Tracker to be scored the same as the Developers were.

Finally, it is recommended that the project selection process begin as early as the semester before students take their senior-level Software Engineering course. This enables students to begin thinking of ideas for possible projects early, in order that they may be better prepared to begin the software development process as soon as the semester begins.

V. Conclusion and Recommendations for Future Work

Young has adopted Needham's [10] practice of learning-by-doing to teach the principles of software engineering using a project-based approach in UCA's CSCI4490 Software Engineering course. He has improved upon this approach by having all members of the class work together as a team on the same project over the duration of the course rather than in smaller groups on a lesser project, as suggested by Coppit [9]. The course project was chosen from a list of student-provided topics, with final selection made based on student preference and a set of criteria provided by the professor. Young followed Needham's method [10] of using a milestone-based approach in developing the project to emphasize the various phases of the software development process and to ensure steady progress was made on the project. The task management system Issue-Tracker [9] was also adopted to encourage equal contribution by everyone in completing the class project and to subsequently determine the level of effort actually put forth by each member of the class. Together, these components contribute to a course that introduces students to the principles of large scale software development while providing them team-oriented experience which should prove invaluable following graduation upon entering the computing workforce.

Together with some of the lessons learned mentioned in the previous section, several Recommendations for Improvement are provided for possible future implementation. First, is the possibility of dividing the course into a one semester course covering Systems Analysis and Design followed by a second course covering Implementation and Testing. Students frequently comment that not enough time is devoted to implementation and testing, so splitting the course up over two semesters might enable their concerns to be addressed without reducing the amount of material which can be covered. While Demurjian and Needham [10] point out some of the pitfalls of splitting the course into two semesters, such a recommendation should be evaluated based on the particular circumstances of the university.

As detailed in earlier sections, the current Software Engineering course is taught using the Unified Process for object-oriented software development. One of the recent trends in software development has been the use of Agile development methods, particularly for non-safety critical systems. It is recommended that the use of Agile methods be investigated for possible future course project development. In addition to introducing students to the use of a different software process, it might also serve as the basis for possible comparison studies between the two methods.

Finally, rather than having students develop a new project from scratch, it is recommended that a future class investigate the modification of an existing project, such as enhancing an existing open source development. As, according to Schach [5], up to 75% of all software development involves maintenance of an existing product, such a project would give students real experience in the type of development effort they are most likely to see upon entering the workforce.

APPENDICES

Appendix A-Project Deliverable Summary

Milestone	**Deliverables**
Milestone 0	Draft Requirements Document
Milestone 1	Requirements Document Rapid Prototype
Milestone 2	Extended Prototype Updated Requirements Document Specification Software Project Management Plan (SPMP)
Milestone 3	Updated Specification Updated SPMP High-level Design Traceability Matrix
Milestone 4	Updated Specification Updated SPMP High-level Design Detailed Design Traceability Matrix
Milestone 5	Updated SPMP Updated Design Implementation Draft User's Manual Traceability Matrix
Milestone 6	Final Requirements Document Final Specification Final SPMP Final Design Final Implementation Final User's Manual Implementation Testing Final Traceability Matrix

Appendix B- Requirements Document Template

(AFTER IEEE 830-1998)

DOCUMENT TITLE

Author(s)
Affiliation
Date

A. Introduction

- **Purpose of document**
  Describe purpose of document, and intended audience.
- **Purpose of project**
  Describe purpose of what software is to accomplish.

B. Background

- **Product Sponsor**
  Provide overview of organization sponsoring development of product.
- **Product Need**
  Provide a brief overview of need for product. Describe essential problem(s) confronted by user community.

C. Project Overview

- **Overview**
  Provide overview of product defined as result of requirements elicitation process. Describe general functionality required of product. Include all system-wide non-functional requirements. (May include "wish list" of desirable characteristics.) Describe how and when users interact with system.
- **(Optional) Similar System Information**
  Describe relationship of product with any other products. Specify if it is intended to be a component of larger product and if so, discuss the relationship.
- **Subsystem breakdown**
  Provide preliminary list of subsystems

D. System Functionality

Provide use case diagrams which capture the functionality of the intended system, including a brief description of each of the functions defining the system as well as a list and short description of each of the intended users of the system.

E. Subsystem Descriptions

From the above use case diagram, identify natural relationships among use cases where the system may be broken up into subsystems; for example a previous semester's Final Exam Scheduler had subsystems for data input and reporting, the scheduling algorithm, and schedule statistics.

1. **Name**
2. **Description**
   Describe subsystem including how the user interacts with it.
3. **Requirements**
   List requirements accomplished by subsystem.
4. **Technical issues**
   Describes any design or implementation issues involved in designing/implementing the subsystem and the respective requirements potentially affected.
5. **Dependencies with other subsystems**
   Describe interactions with other subsystems.

F. Appendices

- **Definitions, Acronyms, Abbreviations**
  Provide definitions of unfamiliar definitions, terms, and acronyms.
- **References**
  Provide citations to all documents referenced or used in preparation of document.

Appendix C Milestone Specification Document

CSCI4490, Fall 2013
Milestone 2 – *Extended Prototype, Object-Oriented Analysis Increment 1*

1. Deliverables:

a. **Extended Prototype:** Extend the rapid prototype delivered in Milestone 1 to provide a prototyped user interface for each of the use cases included in the updated Requirements Document to be delivered as per part 1.b below that define the view your product presents to the actors of that use case. Since this is a Rapid Prototype, your GUI is not expected to invoke the functionality selected by the various buttons or other GUI components; however, if your GUI includes multiple interconnected windows then the functionality for operation of each window and interaction between windows shall be provided.

b. **Updated Requirements Document:** Provide an updated Requirements Document that includes any corrections and clarifications to comments raised during the Milestone 1 Review as well as additional functionality needs discovered during preparations for this milestone. At a minimum, the Milestone 1 Specification Document should be reviewed to address any comments made by the instructor during the Milestone 1 Review. The Specification Document shall be developed in accordance with the included Specification Document Template. Any changes to the requirements contained in this document beyond this milestone shall require written instructor approval using the provided Change Request Form.

c. **Specification Document:** Follow the outline for the eight-part Specification Document provided below in preparing the sections of your Specification Document. In general, your specification document shall cover all aspects of the project; however, the OOA portion of the document shall be developed in two increments over this milestone and the next. Details of the phased development are provided below.

1) **Introduction** (Overview/Problem Statement)

2) **Assumptions:** Use the Assumptions section to document any assumptions you make as your project development continues. For this project, we will use the Assumptions section to prevent you from getting bogged down. If you encounter what you feel are omissions, ambiguities or contradictions in the requirements description, make whatever reasonable assumption you feel will resolve the problem, documenting all such assumptions in the Assumptions section. Make only reasonable assumptions. In case you are unsure of how to proceed in overcoming a requirement description problem, contact the instructor. Note that the Assumptions section is meant to allow you to progress without having to contact the instructor for guidance except in extraordinary cases.

3) **Glossary** (Data Dictionary)

4) **Operating Environment** (Environment in which system shall run)

5) **Interfaces** (GUIs (screen shots) for user interfaces, ER diagrams for Datastores/ Databases)

6) **Object-Oriented Analysis.** In consultation with the instructor, the project manager shall select half of the use cases specified for the project in the Requirements Document for analysis during this milestone. The remaining use cases shall be covered in the next milestone. For the selected use cases, the following shall be provided:

a) **Scenarios:** Provide scenarios (normal/abnormal) which demonstrate the use of each of the top-level use cases selected for analysis during this milestone.

b) **Class modeling:**

(1) Use noun extraction to define the preliminary UML class diagram for your project. This section shall document all stages of noun extraction, including providing a single paragraph describing the software product, identifying the nouns from the description, and eliminating abstract nouns or those outside the problem boundary. Create a preliminary UML class diagram which incorporates the remaining nouns and identifies any relationships among the classes.

(2) For each class identified in 1.c.6)b)(1) above that is required to implement a use case selected for analysis during this increment, create a **CRC card** using the techniques described in the text. With the CRC card collaboration sections, indicate the 'uses' or 'used by' nature of all collaborations.

c) Dynamic modeling (to include UML state chart(s)): For each class that provides a control function used during this first increment, provide a UML state chart that captures the states the class may assume as well as any actions that may be performed within a state or while transitioning between states.

7) **Non-functional requirements:** Provide a description of the non-functional requirements to be met by the system in the following areas:

a) **Performance** (Performance parameters to which system shall conform, min/max #users, timing constraints, etc.)

b) **Parallelism** (Portions of system that execute in parallel)

c) **Concurrent Engineering** (Portions of the system that can be developed in parallel)

d) **Security** (What access shall be controlled, passwords)

8) **Traceability Matrix:** Show how each use case selected from the Requirements Document presented in Milestone 1 for analysis during this milestone is realized in the specification presented during this milestone.

d. **Software Project Management Plan (SPMP):** The SPMP shall be used to manage all aspects of the project development. It is the primary vehicle used by the Project Manager to define and control the work plan for creating the software product. The SPMP has three main components: the work to be done, the resources with which to do it, and the money to pay for it all. Your SPMP shall concentrate on the first two components as your labor is free (from the perspective of this project, anyway!). The SPMP for your project shall cover those use cases selected from the Requirements Document presented in Milestone 1 for analysis during this milestone and shall include the following sections (taken from IEEE Standard 1058):

1 Overview.

1.1 Project summary.

1.1.1 Purpose, scope, and objectives.

1.1.2 Assumptions and constraints. Any assumptions underlying the project, together with constraints such as the delivery date, resources, and artifacts to be reused.

1.1.3 Project deliverables. All the items to be delivered to the client, together with the delivery dates.

1.1.4 Schedule summary. A summary of the overall schedule. See linked Project Schedule Summary

2 Definitions and acronyms.

3 Project organization.

3.1 External interfaces. Provide an overview of the client organization.

3.2 Internal structure. Describe the structure of the development organization itself.

4 Work plan.

4.1 Work activities. Provide a work breakdown structure for the project to include the high-level phases and activities for the entire project, as well as the detailed tasks required to perform the activities of this milestone.

4.2 Schedule allocation. Provide a Gantt or Pert chart capturing all of the phases, activities, and tasks captured in the work breakdown structure provided in section 4.1 above.

5 Control plan

5.1 Requirements control plan. Describe the mechanisms to be used to monitor and control changes to the requirements, specifying for each milestone who is responsible for configuration control of the primary program artifacts (Requirements Document, SPMP, Specification, etc.).

e. **Task Summary**. Team leaders shall provide a detailed breakdown on the tasks completed during this milestone, including task number, brief task description, task area (requirements, analysis, design, implementation, project management), and hours completed, for each member on their team.

2. Presentation: The presentation shall include:

a. **Demo**: A demonstration of your "Extended" prototype software. In your demonstration, show how your prototype meets the specification, and discuss what other areas of the project will need to be completed before your system can meet the scenario in an acceptance testing situation.

b. **Updated Requirements Document.** Provide a presentation of any significant changes to the Requirements Document since Milestone 1. Emphasis should be on any additions, deletions, or modifications to the list of requirements contained in the document and shall include the rationale for the change.

c. **Walkthrough of Specification Document/OOA:** Conduct a walkthrough of the Specification Document focusing on the major functional grouping areas specified in the updated Requirements Document delivered in part 1.b. above. Particular emphasis should be placed on the Object-Oriented Analysis section, covering the items which demonstrate the use of each of the top-level use cases selected for analysis during this milestone. Each team shall be prepared with a list of questions and a list of suspected faults concerning the other teams' functional area to be raised during the walkthrough. The walkthrough presentation shall include:

   1) AT LEAST one of each of the following: **Use-Case diagram, UML Class Diagram, and UML StateChart diagram.**

d. **Work Activities/Schedule Allocation.** Include a slide of the Gantt or Pert chart which captures all of the phases, activities, and tasks included in the work breakdown structure provided in section 4.1 of the Software Project Management Plan (SPMP).

e. **Static GUI Screenshots:** Have slides prepared that show static GUI screen shots of your system in operation that you can use in the event you have difficulty running your live demonstration. **Note that Control-Alt-PrintScreen will let you copy and paste the active window into PowerPoint.**

f. **Copies of Slides:** In addition to the paper copy of the Specification, Software Project Management Plan, Requirements Document, and Acceptance Test Plan from section 1 above, also provide a paper copy of all slides and screen shots used in your oral presentation to your instructor prior to beginning your oral presentation.

Notes:

- Each member shall participate in all portions of the term project, including *each* oral presentation.
- Each team shall be fully ready to go at the beginning of the presentation period to include handing in a paper copy of all slides and GUI screen shots used in the presentation/ software demonstration. Each team shall have 20 - 25 minutes to complete their presentation.
- Any team not providing a paper copy of all deliverables (excluding presentation slides) by the time due, or not providing paper and electronic copies of their presentation by the time due, or not ready to deliver their presentation/demonstration when called upon, shall have 25 points deducted from their milestone grade and shall go to the end of the presentation cycle for that day. Presentations not delivered during class on the due date shall earn a grade of zero, but shall still have to be completed and turned in to receive a passing grade for the course.

Appendix D: Milestone Grading Sheet

Milestone 2/Presentation Grading Sheet

COURSE: CSCI 4490 DATE: _____

PROJECT: _____

Weight	Topic	Score
80%	**Deliverables:** • **Extended Prototype (25%):** prototype of GUI covering all aspects of user interaction for the entire project • **Updated Requirements Document (10%):** Provide an updated version of your requirements document that corrects any issues identified at the Milestone 1 review as well as resolves any discrepancies identified during preparation of the Specification Document • **Specification Document (30%):** o **Introduction/Assumptions/Glossary (3%)** o **Operating Environment/Interfaces(3%)** o **Object-Oriented Analysis (20%)-** includes: 1. **Scenarios-** at least 2 scenarios for each of the top-level use cases 2. **Class modeling** to include noun extraction and corresponding UML Class Diagrams, and CRC Cards 3. **Dynamic modeling** to include UML state-charts for each class which provides a control function for the system o **Non-Functional Requirements (4%)** to include: 1. **Performance/Parallelism/Concurrent Engineering** 2. **Security/Risk Analysis** 3. **Traceability Matrix** • **Software Project Management Plan (SPMP) (15%):** o **Project Overview(1%)-** provide project summary, purpose, scope and objectives, assumptions and constraints, deliverables, and schedule summary o **Project Organization (1%)-** provide overview of client organization and development team structure o **Work Breakdown Structure (WBS) (8%)-** provide WBS through the activity level (level 3) for the entire project with additional tasking to the task level (level 4 and beyond) for the current milestone o **Schedule Allocation (5%)-** provide a Gantt or Pert chart capturing all of the activities comprising the project (as contained in the WBS) as well as all interdependencies among activities.	

20%	**Presentation:** • **Extended Prototype (6%):** prototype of GUI covering all aspects of user interaction for the entire project • **Updated Requirements Document (5%):** Provide a presentation of any significant changes to the Requirements Document since Milestone 1. • **Specification/OOA Walkthrough (5%):** o Presentation team provides walkthrough of the Specification and Object-Oriented Analysis o Other teams provide list of questions and suspected faults o Presentation includes Use Case Diagram, UML Class Diagram, and UML Statechart Diagram • **Work Activities/Schedule Allocation (4%):** Include a slide of the Gantt or Pert chart which captures all of the phases, activities, and tasks included in the work breakdown structure for the entire task, focusing on this milestone.	
	• **Copies of Slides:** did not provide a paper copy of all slides and screen shots used in your oral presentation to your instructor prior to beginning the oral presentation. **(-5%)** • **Team Participation: not all members** of team **involved** in presentation of material **(-5%)** • **Not Prepared to begin** presentation at start of period. **(-25%)** Required copies of slides/documentation/GUI screen shots/presentation grading sheet delivered to instructor at start of presentation	
	Total	

Comments:

Appendix E: Team Leader Grading Sheet

Team Leader Milestone 2 Grading Sheet

COURSE CSCI 4490 DATE: _____

Team Leader: _____

Weight	Topic	Responsibility Number	Score
15%	1. Conduct adequate planning for project execution	4	
10%	2. Effective use of Issue Tracker to manage project completion	4, 5	
15%	3. Adherence to project schedule	5, 6	
15%	4. Quality of project deliverables	5, 8, 9	
15%	5. Equitable assignment of project tasks	2	
15%	6. Effective communication with project manager	1, 6	
15%	7. Provide effective direction to team	3, 7, 10	
	TOTAL		

Comments:

REFERENCES

[1] R. Pressman, *Software Engineering: A Practitioners Approach*, 7th ed., New York, McGraw-Hill, 2009.

[2] I. Sommerville, *Software Engineering*, 9th ed., Boston, Addison-Wesley, 2010.

[3] S. Pfleeger, J. Atlee, *Software Engineering*, 4th ed., Upper Saddle River, NJ, Pearson, 2009.

[4] V. Berzins, Luqi, *Software Engineering with Abstractions*, Reading, MA, Addison-Wesley, 1991.

[5] S. Schach, *Object-Oriented and Classical Software Engineering*, 8th ed., New York, McGraw-Hill, 2010.

[6] M. Prince, "Does Active Learning Work? A Review of the Research," *Journal of Engineering Education*, vol. 93: pp. 223-231. Jul. 2004.

[7] M. Shaw, and J. Tomayko, "Models for Undergraduate Project Courses in Software Engineering," Software Engineering Institute, , Pittsburgh, PA, Tech. Rep. CMU/SEI-91-TR-010, 1991. Available FTP: http://www.sei.cmu.edu/library/abstracts/reports/91tr010.cfm

[8] I. Jacobson, G. Booch, J. Rumbaugh, *The Unified Software Development Process*, Reading, MA, Addison-Wesley, 1999.

[9] D. Coppit, "Implementing Large Projects in Software Engineering Courses," *Computer Science Education*, vol. 16, no. 1, pp. 53-73, Mar. 2006.

[10] S. Demurjian and D. Needham. (2009). Experiences in Project-Based Software Engineering: What Works, What Doesn't. Software Engineering: Effective Teaching and Learning Approaches and Practices. H. Ellis (ed.), IGI Global, pp. 191-211.

[11] D. Coppit, J. Haddox-Schatz, Large Team Projects in Software Engineering Courses, In 2005 Proceedings of ACM Special Interest Group on Computer Science Education (SIGCSE'2005), St. Louis, Mo., Feb. 23-27, 2005, pp.137~141.

[12] *UCA Undergrad. Bulletin 2013-2014*, "Courses in Computer Science", [Online], Available http://uca.edu/ubulletin2013/courses/computer-science/

[13] *USNA Online Viewbook*, "Computer Science Courses," [Online], Available http://www.usna.edu/CS/academics/sitcourses.htm#ic470

[14] J. Selingo, "Skills Gap? Employers and Colleges Point Fingers at Each Other," The Chronicle of Higher Education, [Online], Sept. 12, 2012, Available http://chronicle.com/blogs/next/2012/09/12/skills-gap-employers-and-colleges-point-fingers-at-each-other/

[15] L. Ford, "Graduates Lacking Soft Skills, Employers Warn," Education Guardian [Online], Jan. 30, 2007, Available http://www.guardian.co.uk/ money/2007/jan/30/workandcareers.graduates

[16] *Java EE at a Glance*, [Online], Available: http://www.oracle.com/ technetwork/java/javaee/overview/index.html.

[17] *Amazon.com*, [Online], Available: http://www.amazon.com/.

[18] *Google.com*, [Online], Available: https://www.google.com/.

[19] *Yahoo.com*, [Online], Available: http://www.yahoo.com/.

[20] *NetBeans IDE- The Smarter and Faster Way to Code*, [Online], Available: https://netbeans.org

[21] *Eclipse*, [Online], Available http://www.eclipse.org

[22] *JBuilder- The fastest way to develop enterprise Java™ applications*, [Online], Available: http://www.embarcadero.com/products/jbuilder

[23] G. Booch, J. Rumbaugh, I. Jacobson, *The Unified Modeling Language User Guide*, 2nd ed., Reading, MA, Addison-Wesley, Sep 1, 2005.

[24] *ArgoUML*, [Online], Available: http://argouml.tigris.org/

[25] *Violet UML Editor*, [Online], Available: http://sourceforge.net/projects/ violet/

[26] *Use UML to create Class, Sequence, Use Case, Activity, or State diagrams*, Microsoft Office Visio, [Online], Available: http://office.microsoft.com/en-us/visio-help/use-uml-to-create-class-sequence-use-case-activity-or-state-diagrams-HA102749764.aspx

[27] *Apache Subversion- Enterprise-class centralized version control for the masses*, [Online], Available: http://subversion.apache.org/

[28] *TortoiseSVN- the coolest interface to (Sub)version control*, [Online], Available: http://tortoisesvn.net/

[29] *Git –fast-version-control*, [Online], Available: http://git-scm.com/

[30] *(Microsoft) Project Online*, [Online], Available: http://office.microsoft.com/en-us/project/

[31] J. Hayes , T. Lethbridge , D. Port, Evaluating individual contribution toward group software engineering projects, Proceedings of the 25th International Conference on Software Engineering, May 03-10, 2003, Portland, Oregon.

[32] H. Smith, III, D. Smarkusky, "Competency matrices for peer assessment of individuals in team projects," Proceedings of the 6th conference on Information technology education, Newark, NJ, October 20-22, 2005.

[33] V. Yip, P. Young, P. Marupally, "Evaluation of Methods Used for Measuring Individual Performance on Group Projects," The 2009 Conference on Applied Research in Information Technology, Conway, AR, Feb. 2009.

[34] Gravetter, F.J. and Wallnau, L.B., *Essentials of Statistics for the Behavioral Science*, 7th edition, Wadsworth Publishing, 2010.

[35] Young, P., Yip, V., Lenin, R.B., "Evaluation of Issue-Tracker's Effectiveness for Measuring Individual Performance on Group Projects," 50th ACM Southeast Conference, Tuscaloosa, AL, March 29-31, 2012.

Paul Young received B.S. and M.S. degrees in Computer Science from the University of Mississippi and an M.S. and Ph.D. in Software Engineering from the U.S. Naval Postgraduate School. A career Naval Officer, he served the last five years of his Navy career in the U.S. Naval Academy's Computer Science Department and subsequently transitioned to UCA's Computer Science Department where he teaches Computer Science and Software Engineering. His research interests include software system interoperability, information search and retrieval, the Semantic Web and Health Information Technology systems' interoperability issues.

Donald Needham is a professor of computer science at the United States Naval Academy. His research interests include cyber security, reverse engineering of malware and safety-critical software.