

University of Nebraska - Lincoln

DigitalCommons@University of Nebraska - Lincoln

UNL Faculty Course Portfolios

Peer Review of Teaching Project


2018

Benchmark Portfolio for SOFT 261: Software Engineering IV

Suzette Person

University of Nebraska-Lincoln, sperson@cse.unl.edu

Follow this and additional works at: <http://digitalcommons.unl.edu/prtunl>

 Part of the [Higher Education Commons](#), and the [Higher Education and Teaching Commons](#)

Person, Suzette, "Benchmark Portfolio for SOFT 261: Software Engineering IV" (2018). *UNL Faculty Course Portfolios*. 117.
<http://digitalcommons.unl.edu/prtunl/117>

This Portfolio is brought to you for free and open access by the Peer Review of Teaching Project at DigitalCommons@University of Nebraska - Lincoln. It has been accepted for inclusion in UNL Faculty Course Portfolios by an authorized administrator of DigitalCommons@University of Nebraska - Lincoln.

Peer Review of Teaching Program 2017-2018

Benchmark Portfolio

for

SOFT 261: Software Engineering IV

Spring 2018

Prepared by:

Suzette Person

Department of Computer Science and Engineering

University of Nebraska-Lincoln

suzette.person@unl.edu

Table of Contents

Abstract.....	3
Choosing SOFT 261 for a Peer Review Course Portfolio	4
Background.....	4
Key Goals.....	5
Course Description.....	6
Goals and Objectives of the Course.....	6
Rationale	6
Context.....	8
Enrollment and Demographics	8
Teaching Methods, Course Materials and Outside Activities	8
Module 1 - Methods and Rationale.....	9
Module 2 - Methods and Rationale.....	11
Module 3 - Methods and Rationale.....	12
Course Materials	13
Outside Activities.....	14
The Course and the Broader Curriculum	16
Analysis of Student Learning.....	17
Analysis of Selected Assignments	18
Analysis of Student Perceptions	24
Summary of Planned Changes	26
Summary and Overall Assessment of the PRT Portfolio Process	28
Appendix A.....	29
Appendix B.....	38
Appendix C.....	41
Appendix D.....	44
Appendix E	46
Bibliography	49

Abstract

This benchmark portfolio documents the course objectives, teaching strategies, and assessments for the inaugural offering of SOFT 261: *Software Engineering IV* at the University of Nebraska-Lincoln (UNL). This is the final course in the core sequence of software engineering courses taken by students in the new undergraduate program in software engineering at UNL. These courses teach fundamental computer science concepts in the broader context of engineering software. As an ACE (Achievement-Centered Education) 2 course, the instructional material in SOFT 261 is focused on teaching visual communications skills in the context of applying software engineering processes to a real-world software project. This portfolio describes the course objectives and how this course fits into the broader context of software engineering education at UNL. It also describes the instructional strategies used to teach visual communications embedded in a software engineering course and the assessments used to evaluate student learning. This portfolio also analyzes student learning to assess the effectiveness of the teaching strategies and course materials. Finally, this portfolio reflects on the intellectual challenges of designing and teaching a visual communications course specifically for software engineering majors that incorporates team-based, hands-on learning working with and communicating with software developers on a large open-source project.

Choosing SOFT 261 for a Peer Review Course Portfolio

Background

During their first two years in the software engineering program, students complete four core software engineering courses. These courses were designed following a Software Engineering First (SE-first) model¹, where software engineering concepts are taught early in the program and integrated with core computer science topics to provide a context for learning and applying computing concepts. The alternative model, Computer Science First (CS-first), which is the traditional model for teaching undergraduate software engineering, is focused primarily on teaching computer science concepts during the first two years, followed by two years of primarily software engineering courses. To the best of our knowledge, the UNL software engineering program is the only SE-first program in existence anywhere in the world. Although there is no clear evidence to show one model is better than the other model, we believe that existing undergraduate software engineering programs have chosen the CS-first approach for financial reasons and the ready availability of books and materials, rather than for merits related to student learning. Our decision to choose the SE-first model for UNL's undergraduate program in software engineering was motivated by our teaching experience and our previous experience as practicing software engineers. It was made possible through the support of the university administration. We believe that an SE-first curriculum has the potential to inform students early in their academic studies what a career in software engineering looks like. It also encourages students to think like an engineer from the beginning, learning and practicing the many engineering activities involved in developing and maintaining real-world software systems beyond coding. An SE-first curriculum also has the potential to discourage bad habits (e.g., hacking code together) and to encourage students who may excel at non-coding activities (e.g., design).

Our goal in choosing to build a teaching portfolio for this particular course is to describe our experiences and outcomes in developing an SE-first course that:

- Is primarily focused on communication skills,
- Provides students with experience using disciplined software engineering process models, and
- Enables students to contribute to a real-world software project and communicate with software developers on that project.

As was the case with the other three courses in the software engineering core, there were no models for us to use in the design of this course. This challenged us to think deeply about how we could leverage research-based instructional strategies to teach a course that inter-weaves teaching of visual communication with teaching of software engineering, and that supports student contributions to an open-source project. We also were challenged (and to some extent, guided) by the fact that the course is required to meet specific requirements in order to fulfill the [UNL ACE 2](#) certification requirements. The motivation for developing SOFT 261 as an ACE course is based on our recognition that verbal and visual communication skills are essential for success in the field of software engineering. We believe that by teaching an integrated studies course combining technical and non-technical topics, students can learn to appreciate the value of non-technical skills in their technology field of study.

Although we faced significant challenges in creating this course, we also had several advantages working in our favor. First, we designed and taught the first three courses in the software engineering core, so we were intimately familiar with the software engineering material the students had learned in the previous three semesters. Second, we were well acquainted with the students in the program and their abilities. The students who participated in the inaugural version of SOFT 261 are the first cohort through the software engineering program. They have formed a strong bond with each other as the “test subjects” for our new curriculum and have been willing to provide candid (and valuable) feedback on the course activities and materials for all of the core courses.

Key Goals

Designing a curriculum that is unique in how and when it delivers content presents significant challenges. But at the same time, it also provides tremendous opportunities to think about teaching in new and exciting ways. The primary challenges of designing *Software Engineering IV* include:

- The lack of course materials that integrate computer science, software engineering, and visual communications into a single course,
- A need to create a course that is scalable to handle the rapid growth in the program,
- A desire to create a course that provides opportunities for students who are drawn to, and excel in, the non-coding aspects of software engineering; to provide encouragement and an environment where they can build on their strengths and excel in the field of software engineering, and
- A desire to create a course where students learn communication skills and their importance in software engineering by working with practicing software developers and by contributing to a real-world software system.

My key goals for creating this portfolio were to:

- Apply the Peer Review of Teaching process to create the Software Engineering IV course such that the course objectives, activities and assessments are aligned, and the course:
 - Continues the themes set in the first three courses,
 - Uses backward design² principles and our experiences in teaching the first three core courses,
 - Provides a capstone experience, and
 - Is scalable without diminishing the quality of student learning.
- Create a living document to:
 - Support assessment and refinement of the course over time as we learn what strategies are effective for teaching software engineering and communication skills to students during the first two years of an undergraduate program,
 - Provide a guide to future instructors of the course,
 - Demonstrate the merits of my teaching for reappointment and promotion,
 - Support the ACE 2 certification process, and
 - Provide evidence and supporting information for the dissemination of our experiences and success in teaching an SE-first curriculum.

Course Description

Software Engineering IV (SOFT 261) is a sophomore-level course offered once each year, during the spring semester. It is open only to software engineering majors. The focus of *Software Engineering IV* is on the UNL Achievement Centered Education (ACE) 2d requirements--producing or interpreting visual information. In this course, students learn and practice techniques for creating visualizations to communicate ideas. They also learn visual literacy skills. Both are taught in the context of designing, building, analyzing, and maintaining software using disciplined software development processes and tools to complete a capstone project. Students attend two 75-minute class meetings each week with the instructor(s), and one two-hour lab session each week led by a graduate teaching assistant (TA). The format of the instructor-led class meetings is primarily short interactive lectures followed by guided active learning exercises or team time. Class meetings also include student presentations and guest lectures. Lab sessions are a combination of guided learning activities, in which students practice the application of software engineering concepts, and time for students to work on their capstone project. Attendance at all class meetings and lab sessions is mandatory; unexcused absences result in the student losing attendance points. For the inaugural offering, the course was taught by myself and another professor of practice, Dr. Brady Garvin.

Goals and Objectives of the Course

Software is developed by teams of people, often with diverse backgrounds, skills, and interests. Some team members may have a technical background, while other team members may represent the clients or users who have limited or no technology background. The ability to effectively communicate ideas and concepts to both technical and non-technical audiences is critical for success in software engineering. The primary objectives of *Software Engineering IV* are to prepare students to work individually and in teams to:

1. Visually communicate software engineering concepts to both technical and nontechnical audiences,
2. Formulate and communicate constructive feedback on visualizations and content in peer communications, and
3. Apply disciplined software engineering principles, and recognized practices, to software development and maintenance.

These objectives contribute to [Student Learning Outcomes 1 and 2](#) in the UNL software engineering program, and support the [ABET Student Outcomes \(a\), \(c\)-\(e\), \(g\) and \(k\)](#) and the ABET “Software and Similarly Named Engineering” [program criteria](#).

Rationale

The objectives for this course were chosen based on the requirements for an ACE 2 course at UNL and the importance we placed on teaching non-technical skills during the design of the software engineering program. Below, we elaborate on our rationale for choosing each learning objective.

Objective 1: Visually communicate software engineering concepts to both technical and nontechnical audiences.

In software engineering, as with any discipline that deals with complex systems, diagrams and visual representation are commonly used to communicate ideas when brevity or succinctness is required (e.g., during an oral presentation). In the first year of the software engineering program, students learn how to visually represent information about code using control-flow graphs, call graphs, class diagrams, etc. However, these graphs and diagrams are not useful for representing large, complex systems, or for non-technical audiences. Software engineers also need to be able to communicate ideas at a higher level of abstraction (e.g., at the system level) to both technical and non-technical audiences. Visualizing abstract ideas is hard. It requires the ability to 1) internalize the complex idea or concept in order to identify the key elements necessary to convey the idea, 2) frame the content for the audience by choosing the appropriate terminology, visual idioms, etc., and 3) create the visualization by integrating all of the sub-parts. By learning and practicing the application of these skills using established design principles, students can improve their ability to communicate complex ideas and concepts. They can also improve their confidence in working with diverse audiences and overcome a common misconception that visual communication requires artistic talent.

Objective 2: Formulate and communicate constructive feedback on visualizations and content in peer communications.

By practicing formal and informal reviews of peers' work, students learn how to interpret visual information (e.g., visual literacy skills) while also discovering the diverse ways in which ideas can be represented visually. Students also learn and apply established metrics for evaluating communication artifacts, and they practice critical thinking skills by providing constructive, specific, and actionable feedback (positive and negative) to their peers. Through this form of peer learning, students have the opportunity to observe and learn from other students how (and how not to) communicate visual information.

Objective 3: Apply disciplined software engineering principles, and recognized practices, to software development and maintenance.

Throughout the core software engineering courses, students learn that software engineering is much more than programming (i.e., writing code). Software engineers spend a considerable amount of time on non-programming tasks including researching ways to solve problems and studying code to understand how it works, how it can be changed, and to locate errors in the code. They also plan how they will change the code and how they will test their changes, and they spend time meeting with clients and team members to talk about the software and to discuss the status of the software. This wide range of activities relies not only on strong technical skills, but also on strong teamwork, time management, planning, and communication skills. However, in the first three core courses, software engineering majors are primarily focused on learning foundational technical knowledge and skills, resorting to ad hoc processes to facilitate teamwork and communication. This lack of instruction in these “soft” skills during the first three semesters provides the students with multiple opportunities to experience first-hand, the risks and impact of working without structured processes and good communication skills. By the end of SOFT 261,

students will have worked on two capstone assignments providing numerous opportunities to learn and appreciate the value and impact of the rigorous software development processes and communication skills taught in the course.

Context

The UNL software engineering major was launched in Fall 2016 when SOFT 160 and SOFT 161 were offered for the first time. SOFT 260 and SOFT 261 (the course presented in this portfolio) were first offered in Fall 2017 and Spring 2018 respectively, when the first cohort of majors entered the second year of their program. The software engineering major is one of three majors offered by the Department of Computer Science and Engineering at the University of Nebraska-Lincoln. It was developed in response to the increasing demand for software engineers both locally and nationally. It was made possible due to the availability of a top-ranked software engineering research faculty. The software engineering major is offered through the UNL College of Engineering and requires students to complete 124 credit hours of study, including a required internship. Once the program is fully established, the Department will seek accreditation from [ABET](#).

This course (*Software Engineering IV*) fits into the overall software engineering undergraduate curriculum as the fourth, and final course in the core course sequence. At the end of *Software Engineering IV*, students are expected to have the technical and non-technical skills and knowledge to succeed in upper-level courses in both software engineering and computer science. They are also expected to be prepared for their two, year-long capstone experiences in which they work with students in other majors on projects sponsored by members of industry.

Enrollment and Demographics

Students in the inaugural offering of SOFT 261 are the first cohort of software engineering majors at UNL. Because the software engineering program follows a cohort model, the majority of the students in SOFT 261 have studied software engineering together for the previous three semesters (although a small number of students joined the cohort in the third semester after taking a bridge course). The students have previously worked in instructor-assigned teams on courses projects and in randomly assigned pairs during labs in the previous three core courses. In the first offering of the course, we started with 19 students in a single section (18 students completed the course). In Spring 2019 we anticipate the course will be offered to 40-45 students split into two sections. Once the major is fully established, we expect to offer this course each spring to two or more sections of 40-45 students each.

Teaching Methods, Course Materials and Outside Activities

SOFT 261 is organized into three modules. In each module, we utilize a combination of peer learning (e.g., think-pair-share³), in-class activities working in small teams (2-4 students), guided lab activities, and interactive lectures. We also use class time for student presentations and for guest lectures (e.g., Software Engineering in Practice (SEIP) and Software Engineering in Research (SEIR) presentations). All three modules also use journal assignments, outside

activities, and assessments. The daily learning objectives are posted along with the assignments on the course website. The course syllabus and schedule are included in [Appendix A](#).

The choices made in the selection of teaching methods, materials and activities were made based on our experiences in teaching the previous three software engineering courses. We specifically chose to:

- Continue with the same basic approach to teaching software engineering, but with less structure in order to prepare students for their capstone course,
- Create a project-based course where students apply what they have learned in the previous three core software engineering courses, but replace ad-hoc processes with structured processes that leverage established best practices,
- Continue to use SEIP an SEIR presentations to expose students to how the material they are learning is applied in practice and in research, and
- Develop a course that enables students to contribute to an open source project and work with real-world software developers.

Module 1 - Methods and Rationale

The first module is the course introduction and covers the first two weeks of the semester. In this module the students are introduced to the basic components of effective communication, including visual communication. We also introduce disciplined software process methodologies. These methodologies enable development of large complex software systems and facilitate communication between team members and between the developers and stakeholders. Our motivation for exposing students to all of the main course topics in the first module is to highlight the underlying relationships between topics and to motivate the importance of communication skills in software engineering.

For most class sessions in the first module, our approach to teaching is to introduce the topic for the day through a brief interactive lecture at the beginning of class. Student participation is facilitated through the use of index cards to call on students to answer pre-planned (or spontaneous) questions and prompts. Cards are created during the initial class meeting when each student writes his or her preferred name on an index card provided by the instructors. The instructor then brings the cards to each class meeting and calls on the student whose name is on the top of the deck (we occasionally shuffle the cards). The number of students called on during a class meeting depends on the length of the lecture and the number of questions posed to the students. The cards can also be used for taking attendance (we write a tally mark or a date the student is absent or late) and for assigning pairs or teams. Students are free to raise their hand to ask questions or make comments during the lecture, however, questions posed by the instructor are answered by calling on one or more students using the note cards, rather than asking for volunteers to answer a question. This approach to class participation is used in all of the core software engineering courses. It provides a mechanism to engage all students in the discussion without bias. Student feedback on the use of cards indicates it helps them remain engaged during class and also encourages them to come to class prepared knowing that they may be called upon to answer a question during class. During lectures, we also use a think-pair-share technique to encourage students to explore and share their own ideas on a topic with each other prior to sharing with the class as a whole.

In-class activities are typically performed in assigned pairs or small groups. These guided activities include instructions and discussion questions provided by the instructors. Some activities involve students sharing what they have learned from the activity with the rest of the class. To complete an activity, students are expected to use the resource(s) provided by the instructors, locate resources on the Internet, and to draw on their experiences in the previous three semesters of software engineering courses. For instance, in the first class meeting each student pair is assigned to research a communication skill relevant to software engineering and use the [Google slide template](#) provided by the instructor to record their answers to three prompts “When is the communication skill important in software engineering?”, “Why is the communication skill important in software engineering?” and “What does the communication skill look like when done well?”. At the end of class, each pair of students provides a brief (2 minute) summary of their assigned communication skill.

Our rationale for using brief interactive lectures followed by hands-on activities is three-fold: 1) to *encourage students to become independent learners* by making them share the responsibility for their learning, rather than taking the role of passive learner and expecting the instructor to provide all of the information, 2) to *promote peer teaching and mentoring*, a skill that is widely used by practicing software engineers and has also been shown to be an effective learning technique for students, and 3) to *provide regular communication skills practice* by requiring students to solve problems as a team and report back to the class with their solutions. After teaching SOFT 261 using this approach, we have found that this combination of interactive lecture and in-class activity is engaging for both the students and the instructors, and it also enables us, as instructors, to better understand the capabilities of students in terms of independent and peer learning—information that we are using to improve the course.

In the two lab sessions in this module (taught by a graduate teaching assistant), students work in their assigned project teams to set up tools and to research technologies they will use to complete the capstone project in modules 2 and 3. They also begin developing the proposal for their capstone Phase I project. For this assignment, each team of students designs and develops software that builds on the open source project specified by the instructors. Unlike the highly structured lab instructions in the previous three software engineering core courses, the lab instructions for SOFT 261 (provided by the instructors) are much less specific in *how* to accomplish each task. The lack of specificity challenges students to think critically about how to solve problems posed in the lab. The lack of structure forces the students to practice time management skills in order to complete all of the tasks. Our rationale for providing less structured labs than previous semesters is to provide a model of software engineering that more closely resembles the real-world, while still providing the students with a general framework for achieving the learning objectives.

The in-class activities and lab activities in module 1 are primarily intended to provide active learning opportunities that reinforce the concepts presented during the interactive lectures. Students also practice important skills, such as collaboration and communication. These activities also include formative assessments that provide students with real-time feedback, and instructors with insight into student learning. For instance, artifacts created by the students during class (e.g., the slides linking communication skill and software engineering created on the

first day) are reviewed by the instructors for accuracy, misconceptions, etc. and the findings integrated into a subsequent class or activity and used to inform changes to the next version of the course. Lab checkpoints also serve as formative assessments, enabling the lab teaching assistants to check student learning at pre-defined points in the lab and to provide feedback and [Just-In Time Teaching \(JiTT\)](#) instruction when necessary. The outside class activities in this module include reading assignments and journal assignments as shown in the course syllabus in [Appendix A](#).

Module 2 - Methods and Rationale

In the second module (lasting approximately five weeks), students begin to apply visual literacy skills and software engineering processes and tools. Students work in instructor-assigned teams of four students to build software based on a large open source project. In the inaugural offering of SOFT 261, the students worked on [OpenMRS](#), an open source medical records system that they had been working with in previous software engineering core courses. During this module, student learn and practice an Agile software development process widely used in industry ([Course Objective 3](#)). Students track and report progress using an [on-line project management tool](#) that supports Agile software development. Intra-team communication and communication with the instructors and TAs is through on on-line communications tool, [Slack](#). At the beginning of each lab and once a week in class, students also provide brief oral status updates to their team members through a [stand-up meeting](#).

During this module, students also learn basic visualization concepts and a structured process for turning an idea into a visualization that effectively communicates that idea ([Course Objective 1](#)). Students practice applying the process to the development of a visualization that describes the architecture of the software they have developed. At the end of the module, student teams peer-review their architecture diagrams as an in-class activity ([Course Objective 2](#)) and use the input from the peer review to prepare to the final version of the diagrams. The diagrams are then used in a project hand-off presentation to the class.

Teaching methods in this module are relatively the same as module 1. Interactive lectures are used at the beginning of a class session and hands-on activities fill the remainder of the class session. Students also attend a weekly lab session with the teaching assistants to work on their capstone assignment. At the end of the module, two days are used for team presentations and one day of the module is used for an SEIP talk. During the inaugural offering of SOFT 261, the SEIP talk focused on the importance of architecture and the value of the Agile software development process in helping manage problem complexity. In addition to lab time, students have several class sessions for team time—most sessions are guided activities intended to help them complete their capstone activities (e.g., create a draft of their presentation). During this module, many of the journal assignments ask the students to reflect on their capstone experiences, relating it back to the reading assignments in “What Makes a Great Engineer.” During this phase, we also used short quizzes during three lab sessions as a formative assessment of software engineering concepts. In previous software engineering courses, lab quizzes with 2-3 short-answer questions were administered across the semester. On each quiz students practiced the application of concepts recently taught in class. Quizzes toward the end of the semester were also used to help the students review for a cumulative final exam. In SOFT 261, the format of the questions was

changed to multiple choice and multiple true-false with the intention of simplifying the grading. Students in SOFT 261 did very poorly on these assessments and the quizzes were dropped from the students' grade completely. We analyze why students performed poorly in the Section "[Analysis of Student Learning](#)."

Teaching how, when and how often to communicate software status information is challenging. Company policies, practices and procedures vary greatly. Software is always changing. Software systems are huge and complex. All of these factors impact how software engineers communicate. In this module students also learn technologies and tools related to developing and managing software. They experience first-hand, the importance of planning their work and practice time management skills. Although we assigned readings from various sources on the Internet, we were able to find mostly very general information, so we relied heavily on guided hands-on activities and the capstone assignment to teach this module. To offset the risks of hands-on learning, we used class time to deliver JiTT instruction when we observed students struggling either with technology, communication, or process issues. For instance, students struggled to learn the MVC architecture model used by OpenMRS, and therefore had difficulty developing a module. After recognizing this issue, we developed an in-class lecture to help the students learn the architecture. In another instance, we noticed that the students were not applying the visualization process we taught in class. Instead, they were applying ad-hoc processes that omitted many of the planning steps or omitted steps that leverage established visual communications practices. Following this observation, we created an in-class activity that included checkpoints for the instructors to evaluate the application of the process in addition to the end result (i.e., visualization). Although we believe the methods selected for delivering the course material were effective, we also believe that students need more instruction, particularly instruction they can later reference, since most students did not appear to take notes during class (we do not know why this is the case).

Module 3 - Methods and Rationale

In the third module (lasting approximately seven weeks), the students are assigned to new teams of three students each (four, if necessary to balance the teams). The student teams work with the same open source project to perform software maintenance tasks that extend their communication practices to involve the project developers. For this "maintenance" phase of the project, each team is focused on locating and performing one documentation task, one bug fix or new feature task, and one testing task for the open source. The students use the open source project's issue tracker, continuous integration server results, the various sub-projects' GitHub activity information, and the project's website and wiki pages to locate tasks. Students ask for clarification and assistance from the project's developers and explain their ideas and proposals through on-line forums, issue tracker comments, and pull requests (i.e., a communication mechanism for specifying information about a proposed software change). Students use an Agile software process model ([Course Objective 3](#)), and again track progress using an [on-line project management tool](#). Intra-team communication and communication with the instructors and TAs is through an on-line communications tool, [Slack](#).

During Module 3, students also continue to practice visual communication skills ([Course Objective 1](#)) by creating visualizations that document their contributions to the open source

project. Each student presents his or her visualization in an oral project status report to fulfill of Homework Assignment 3.6. The student teams also create several visualizations for use in their in-class “release meeting” presentation at the end of the semester. To meet [Course Objective 2](#), students peer review the visualizations during an in-class exercise. Each student also provides a written evaluation of and feedback on the team presentations and visualizations using a rubric provided by the instructors (including a self-evaluation). For the SOFT 261 final exam, students attend presentations by students in the year-long capstone course and provide a written evaluation and feedback for two presentations. They also create a new visualization or a modify a visualization for one of the presentations to help improve how the information is communicated in the presentation.

Our teaching methods in this module are primarily hands-on activities. Three class sessions were used for individual student presentations. While each student presented a status report to his or her team, the instructors and the TAs, the rest of the teams had time to work on their projects. These oral status reports enabled the instructors to provide feedback to the teams on their projects and to answer questions from the team, while also allowing the instructors to assess individual student’s visualization and communication skills. The last two class meetings were dedicated to team presentations. One class session was dedicated to a Software Engineering in Research (SEIR) talk, and another class session was used for a Software Engineering in Practice (SEIP) presentation. The SEIP talk focused on communicating visually on the white board. One notable point made by this speaker that several students commented on in their journals is the fact that it is not necessary to be an artist to create effective visualizations. In addition to the capstone assignment, students continued to maintain journals during this module, and they completed two homework and two take home exams during this module.

Our rationale for the teaching methods in this module is similar to the previous module—provide students with hands-on experience working on a large-scale software system while providing minimal structure and support. We also want the students to learn how to communicate with real-world software developers. This introduces new challenges in that project team members are from all over the world. Students learn the impact of timing on their communications, and the need to fully describe the problem or issue, to reduce the number of information exchanges and therefore the amount of time waiting to resolve an issue. They also learn that they are responsible for creating a context for their communications—the open source software is so large and so complex, the developers do not retain every detail of the software in their memory and therefore need to be educated or reminded of the details on the part of the software where the students are working. During the inaugural offering of the course, we realized we need to explicitly teach these ideas because students learned them by trial and error which caused some teams to have problems finishing their tasks—this was not our intention. We also recognize that the number of assessments in this module is too high and some of the feedback comes too late.

Course Materials

Course materials that contribute to student achievement of the course learning outcomes include:

- Class lectures
- In-class activity worksheets

- Weekly lab assignments
- Course website
- Piazza
- Journal questions
- Homework assignments
- Capstone project assignments
- Presentation rubrics
- 360 review form
- Quizzes
- Exams

In SOFT 261, the majority of class lectures are brief (approximately 15-20 minutes out of the 75-minute class meeting) followed by an in-class activity. During class lectures, information is generally presented on the whiteboard. Students are expected to take notes (i.e., lecture notes are *not* made available). In-class worksheets are on-line (typically provided as a Google doc). One member of each student groups makes a copy of the worksheet and shares it with his or her team members and the instructors. Worksheets typically include instructions for completing the exercise, space to respond to questions or prompts, and multiple checkpoints indicating when the students are required to share their work with an instructor for signoff before continuing. Students can also use the worksheets as a guide on their homework assignments. Weekly lab assignments include learning objectives, a series of activities to be completed during the lab, links to resources, and multiple checkpoints when the students are required to share their work or status with a teaching assistant before continuing.

SOFT 261 student journal questions and prompts are posted weekly on the course website. Questions cover concepts and material covered in class, lab, or in reading assignments, and reflective questions related to the students' learning goals and achievements and their experiences on the capstone project. [Piazza](#) (an on-line Q&A forum) is used to post announcements and for students to post questions about the course and assignments. SOFT 261 homework assignments assess students' ability to independently apply concepts learned in class. These assignments include learning objectives (based on course objectives [1](#) and [2](#)), detailed instructions for completing and submitting the assignment, and a detailed breakdown of how points are assigned. The capstone assignments are team-based activities that provide students with an opportunity to work in small teams to practice all of the course objectives in an integrated manner. These assignments also provide high-level instructions (what versus how) for completing the assignment and a detailed breakdown of how points are assigned. The presentation rubrics provide guidelines for how the instructors and students evaluate the student presentations, and the 360 review form provides instructions and criteria the students use to evaluate their own contributions and the contributions of their team members at the end of each phase of the capstone project. Finally, the course quizzes assess the students' knowledge of the software engineering concepts taught in the course, and the two exams (mid-term and final) assess students' ability to visually communicate software engineering concepts and to formulate and communicate constructive feedback on visualizations and content in peers' communications.

Outside Activities

Students in SOFT 261 are expected to spend 8-12 hours each week on outside class activities including individual homework assignments, take home exams, weekly journal assignments, and team time spent working on the capstone project. Students are assigned a small number of reading assignments (from sources available on the Internet) to complete outside of class. They are also expected to research and independently learn the technologies necessary to complete their capstone projects.

Homework assignments and take-home exams provide students with formative and summative assessment opportunities to demonstrate their ability to work independently to 1) apply the visual communication development process taught in class, 2) create communications that effectively use the elements of visual communication to convey information, and 3) demonstrate their ability to provide constructive, specific, and actionable feedback on communications created by other students. The rationale for using homework and take-home exams for formative and summative assessments in SOFT 261 (versus in-class assessments) is based on our observations that students find the creative aspects of designing and developing visual communications daunting and often require multiple iterations or multiple attempts to complete an assignment (i.e., requiring more time than would be available in a single class or lab session). We also prefer to use class time for guided activities and to observe student performance as they apply the software engineering and communication knowledge and skills.

Weekly journal assignments are used to guide student reading by providing study questions, assess student understanding of material covered in class and in assigned readings, and to provide students with an opportunity to reflect on their software engineering experience and what they are learning in the course. Journal assignments are included in all of the software engineering core courses. Journal assignments in SOFT 261 contain fewer concept questions than previous semesters and instead include more opportunities for students to reflect on their project experience and on the traits of a great software engineer (based on their reading of “What Makes a Great Software Engineer”⁴). Our rationale for assigning journal questions as an outside class activity is two-fold: 1) journals assignments provide a low-stakes formative assessment opportunity for students to practice answering concepts questions and practice written communication skills, and 2) journal answers provide instructors with key insights into areas where students may be struggling, provide a one-on-one communication channel between the student and the instructors, and inform instructors on students’ perceptions of the course and their learning accomplishments.

Outside of class, students may also work on their capstone project assignments. Although some amount of class time is set aside for teams to work on their projects beginning in middle of the semester, the majority of the capstone work is performed in the weekly lab sessions or outside of class. Due to the variability in the nature of the tasks, task difficulty, team dynamics, etc., some teams may need to spend only a few hours outside of class and lab time working on their project, while other teams may need to spend considerably more time working independently or together on the project outside of class. The rationale for expecting students to work outside of class and lab on the capstone project is that this unstructured work time provides students with additional opportunities to practice communication and software engineering skills in a less structured environment that more closely models the real-world.

The Course and the Broader Curriculum

The UNL software engineering major is offered through the College of Engineering and requires students to complete 124 credit hours of study, including a required internship. After completion of SOFT 261, students take four advanced software engineering courses, 15 hours of technical electives, and two years of a year-long capstone course. Once the program is fully developed, the Department will seek accreditation from [ABET](#).

Software Engineering IV is open only to software engineering majors who have achieved a grade of C+ or higher in each of the previous three core software engineering courses. It is primarily intended to fulfill the students' Achievement Centered Education (ACE) 2 requirement in the context of engineering software. The ACE 2 requirements state that students will

“Demonstrate competence in communication skills in one or more of the following ways:

- a. by making oral presentations with supporting materials,
- b. by leading and participating in problem-solving teams,
- c. by employing communication skills for developing and maintaining professional and personal relationships, or
- d. by producing and/or interpreting visual information.”

Although the course was designed to specifically address the ACE 2(d) requirement, students practice all four components of ACE 2.

SOFT 261 was designed to continue the theme established in the first three core courses of teaching an integrated software engineering and computer science curriculum. To the best of our knowledge, the UNL software engineering program is the only program in existence anywhere that follows the SE-first model of teaching software engineering concepts from the beginning. Our choice of methods, material and activities for SOFT 261 assume students have learned fundamental software engineering and computer science concepts. We also assume students have experience developing software in teams. At the end of *Software Engineering IV*, students are expected to have the technical and non-technical skills and knowledge to succeed in upper-level courses in both software engineering and computer science. They are also expected to be prepared for their two, year-long capstone courses in which they work with students from other majors in the department on team projects sponsored by members of industry.

In the long term, we believe that teaching an SE-first curriculum will impact how students approach software development. First, we believe that four semesters of applying software engineering practices and tools, working on large scale software, working in teams, and learning communication skills in the context of software engineering will enable and encourage the students to solve computational problems with an engineering mindset. Students will be equipped to apply these skills in advanced courses, in their capstone course, in their internships, and in their careers post-graduation. Second, students who prefer the non-programming aspects of software engineering (e.g., design, testing and analysis) will be exposed to those areas of software engineering early in their academic careers and may be more inclined to stay in this field of study. And third, we believe that by learning the value of good design and analysis, and

the importance of writing high quality software, students will create software that is secure and maintainable.

Analysis of Student Learning

Students in software engineering progress through the program as a cohort. At the beginning of SOFT 261, the students have very similar computer science and software engineering background knowledge because they have studied together for the previous three semesters (with the exception of the small number of students who attend the bridge course between the second and third semesters). Furthermore, the faculty who taught the inaugural offering of SOFT 261 are the same instructors who taught the students in the previous three core software engineering courses and the bridge course. This consistency in the student population and academic history provided us with several advantages when writing the course 1) we were able to make certain assumptions about the students' technical knowledge base when deciding on the capstone assignment, 2) we had a collection of teaching methods that the students were familiar with and had helped shape through their feedback in earlier courses, and 3) the student cohort was small and the students knew each other—even if they had not worked together on a team previously, they had seen each in class or lab so they were familiar with each other. Another important advantage we had was the relationship we had established with the first cohort of students. They know they are helping to shape the software engineering curriculum and how it is delivered. They also know that if something does not go well (e.g., the quizzes in SOFT 261—see below), their grades will not be penalized for it.

The majority of the assessed course work in SOFT 261 is performed in teams of three or four students. This work accounts for 55% of the students' grades. Three homework assignments, two take-home exams, several quizzes, and weekly journals facilitate individual assessment of the learning objectives.

The quizzes used in SOFT 261 were ultimately dropped from the computation of the students' final grades. The highest average score across the three quizzes was 79% and the lowest average score was 37%. The quizzes were originally planned to account for 10% of the students' final grades. We updated the weight of the quiz scores towards the end of the semester, reducing it to 5%, but when we saw the impact on the students' final grades, we dropped the quizzes completely. Our rationale was that we could not confidently conclude that they accurately reflected student learning. Although the scores were low, we felt that the format of the quizzes (multiple choice and multiple true/false—formats we had not previously used) and the fact that we had relied heavily on independent learning of the concepts early in the semester indicated that the quizzes may not have been fair. *In future offerings of this course, we plan to provide more instruction and more formative assessments on concepts early in the semester. We also plan to learn how to better use these assessment techniques to confidently assess student learning.*

Journal grades in SOFT 261 account for 5% of the students' final grades. Journals are assigned at the beginning of the week and due at the beginning of the following week. To record their journal entries, students create a Google doc that is shared with the instructors. The journal scores for the semester ranged from 3.85% to 100%. Approximately one-third of the students received 50% or less, one third scored 100%, and the other third scored between 61.5% and

96.15%. Each journal assignment consists of 4-6 prompts. The entire assignment is worth a maximum of two points; one point for effort and one point for professional writing. The correctness of the responses is not considered. Journals assignments are used in all of the core software engineering courses. In SOFT 261, the journal assignments tend to have more reflective prompts (versus writing about concepts taught in class). In all of the software engineering core courses, the journal entries are used for students to specify personal learning objectives for the semester and to indicate where they expect to be challenged. The instructors view the journals as a private communication link with the students and as a mechanism for assessing student learning. The instructors record a comment in the Google Doc, providing feedback and a score. Feedback includes brief comments providing clarification of a topic, encouragement to look a resource to rethink their answer, or just an encouraging thought such as “Looks good!”. In SOFT 261, regular journal entries were submitted by 15 of the 18 students until the middle of the semester, but towards the end of the semester, this number had dropped to approximately 13 of 18. Some students indicated they forgot about the assignments (the assignments are posted weekly on the course website for all of the core software engineering courses). Other students indicated they did not help their learning and therefore did not feel motivated to complete them. Students also noted in passing comments that they had a lot of projects in their courses this semester. *We still believe that reflection is a valuable teaching method and plan to explore ways to help motivate students to use their journals as a learning opportunity.*

Analysis of Selected Assignments

This course has three primary goals. The first two are focused on communication skills in the context of software engineering. These learning objectives can be thought of as foundational and focused. The [first learning objective](#) targets the basic skills related to creating a visualization to represent an abstract software engineering concept or idea. The [second learning objective](#) targets the basic skills related to providing constructive, specific and actionable feedback on visualizations and content in a peer communication of software engineering ideas. The [third learning objective](#) addresses the integration of communication and software engineering skills through the use of communication techniques and tools, along with software engineering practices and methods, to engineer software for a real-world software system. In this section, we describe a subset of the assignments used to assess student learning.

Learning Objective 1

To assess the [first learning objective](#), we assigned Homework 1.4 at the beginning of the second week of class. Students were given one week to complete the assignment. This formative assessment asked students to create a *features matrix* to compare and contrast the features of the software process models they were learning in the course. They were also assigned to write two directed paraphrasings. Each paraphrasing provided the students with an opportunity to restate his or her understanding of the software process models in two contexts and for two different audiences. For the first paraphrasing, students were to write what they would say to a manager who is considering changing the team’s software process model. In the second paraphrasing, the students were to write what they would say to a junior developer regarding how to adapt to the team’s process model and whether he or she should try to introduce agile processes. The primary

objectives of the assignment were to assess the students' ability to communicate their understanding of the software process models and to assess their ability to represent information using a basic visualization technique (a features matrix). [Appendix B](#) contains two examples of student work submitted for Homework 1.4.

The Homework 1.4 assignment includes the definition of a features matrix and instructions for creating the features matrix. Although most students were able to successfully create the matrix layout (headings and labels in a grid fashion), they often chose labels that were ambiguous or lacked sufficient detail to understand the concrete idea represented by the label. For example, in [Appendix B](#), the sample labeled Student A uses “Lengthy” and “Well Documented” as features of the processes. In other student submissions, we also observed labels such as “Flexible,” “Great Documentation,” “Risk Mitigation,” “Manageability,” and “Documentation.” These labels do not articulate a specific feature of a software process model and therefore do not enable the student to explain the differences and similarities between the software process models. This type of error was common across the work submitted by the students. Some students also chose features that do not help the reader distinguish between process models. Either students did not clearly understand the differences and similarities or they were unable to clearly articulate the them using a features matrix (or both). Although many students performed poorly on this assignment, several students were able to create a features matrix using labels that were somewhat better than the labels used by Student A (e.g., Assignment 1.4 from Student B in [Appendix B](#)).

In the second part of the assignment (the directed paraphrasings), many students lost points on the assignment due to basic writing mechanics (e.g., incorrect grammar, punctuation, etc.). Many students also had difficulty applying their understanding of the models to write a brief informative composition to a specific audience. The students also struggled to write persuasively (e.g., to explain why one model is superior to another model). And, in some instances, students wrote the paraphrasings as a stream of facts, rather than structuring the information to create a coherent and connected set of ideas. One thing that surprised us was the conversational nature of the paraphrasings submitted by several students (e.g., Assignment 1.4 from Student C in [Appendix B](#)); we were expecting a paraphrasing that reflected a professionally written statement. We believe the wording of the assignment “write what you would say to...” was confusing to the students and changed this wording for Homework 3.8 (discussed below). *From this assignment, we learned that we need to be more careful in setting the expectations for an assignment and we need to provide more basic instruction on communicating visually than we had originally expected. We also believe that students struggled with the assignment because they did not really know or understand the software models. In future offerings of this course, we will need to provide more instruction on the software process models as well as how to visually represent information, rather than expect the students to self-learn and peer-instruct on this material.*

Due to the low scores on the first homework assignment, we assigned a similar assignment in the third module, Homework 3.8. For this assignment, students created a features matrix comparing three Agile software process models, two of which are models they used in class (Scrum and Scrumban). The third model, Kanban, is very similar to a model they used in class. Students were also asked to summarize the information in the matrix, focusing on the key differences and practical implications. Additional instruction was provided to encourage the students to structure

the description in way that avoids writing just a stream of facts. This assignment was submitted after spring break, so the students had instruction in visual communications and practice using the Scrum and Scrumban versions of an Agile software process model on the two capstone assignments in the course. We also provided another example of a features matrix in the assignment, and students had instructor feedback from the first assignment.

Overall, student performance on Homework 3.8 was much better than on Homework 1.4. To compare the differences in the features matrices created at the beginning of the semester with the features matrices created at the end of the semester, consider the examples of student work in [Appendix B](#) between Homework 1.4 and Homework 3.8. In the first example, Student A includes more descriptive feature labels and more descriptive cell entries in the features matrix in the second assignment. This was true of most students' second submission. In Student B's second features matrix, the terminology and features chosen for the matrix are more specific and are relevant to a comparison of the three models, whereas the labels used in the first matrix are ambiguous and difficult to use in assessing if the student understands the process models and their differences and similarities. *The results of this assignment reinforced our observations from Homework 1.4. We also believe that giving the students "good" examples to use as a model, along with a rubric by which they can evaluate their work would be helpful (unfortunately the ACE 2 rubric is too generic).*

To assess the students' ability to create a more complex visualization, we assigned a take-home midterm that asked the students to create an on-boarding process for an open-source project and to visually represent their process. We also asked the students to justify why their proposed process would benefit new developers, citing their experiences and lessons learned. Students gained on-boarding experience in the capstone project assignments, so they had first-hand knowledge of how to onboard (join) a new project. We also assessed their ability to apply the visualization process we taught in class. This process leverages established visualization design practices to guide the creation of a visualization. Four out of 18 students received full credit for process execution. Students who did not receive full credit lost points for failing to document steps in the process. Many students lost points in the category of visualization content for failing to include all of the process components specified in the assignment. Students also lost points for professional writing in their justification, and for failing to argue concretely for their proposed process. With respect to visualization quality (e.g., effective use of hierarchy, grouping, sequence, position, color, size, shape, orientation, appropriate level of abstraction, creativity and professional writing), most students scored 7 or 8 out of 10 points; all but one student received both points for creativity. The student who lost points for creativity turned in a visualization that appeared to lack any real effort to create an image of the process. [Appendix C](#) includes examples of visualizations illustrating "A", "B" and "C" level work (based on the visualization scores only). The visualization receiving a grade of "C" failed to cover all of the required content and lost points for quality related to effective use of hierarchy and size, and for professional writing (improper capitalization). The visualization receiving a grade of "B" received full credit for visualization quality but lost points for failing to cover all of the required content. The visualization receiving a grade of "A" received full credit for content and visualization quality. We were surprised that students lost points for failing to apply the visualization process and for failing to include all of the required components in the onboarding process. We attribute some of these issues to students rushing through the assignment. Given the amount of instruction

provided and the limited number of formative assessments, we believe the students adequately accomplished this learning objective. *However, for future offerings of this course, we plan to explore instructional techniques for better teaching visual communications and to provide more formative assessment opportunities for students to practice using the process. We will also assign these progressively more challenging visualizations earlier in the semester.*

Learning Objective 2

We assess the [second learning objective](#) in SOFT 261 by first several informal peer reviews during class that are observed by the instructors, but not graded. We assess their ability to perform formal reviews on an individual basis as part of the capstone assignments and on the final exam.

Informal Reviews. In previous software engineering courses, students review their peers' designs, code, and contributions to course projects. In SOFT 261, we built on this experience by asking students to work in teams to perform informal peer reviews of visualizations and presentations created by other teams. The results of these reviews are used to help the teams prepare their capstone presentations. To help students prepare to solicit feedback, they first complete an exercise that guides them through a process to identify feedback that would be useful and that helps them develop questions they can ask to assess the reviewers' understanding of the artifact under review. The first steps in the process are to have the students identify the stakeholders for the artifact under review (e.g., diagram), and then to create use cases from the perspective of that stakeholder. The students then create one or more scenarios for each use case and then use the scenarios to formulate questions that could be used to determine if the artifact supports the scenario.

The peer review process is performed in two rounds during a class session and facilitated using the questions developed in the previous exercise and a peer review worksheet provided by the instructors. Instructors pair the teams. The members of each team divide into presenters and reviewers. After completing the first round of and recording the feedback, students switch roles and perform another round of reviews so that every team member has an opportunity to be both a presenter and a reviewer. Informal peer reviews not only enable the students to practice giving constructive criticism, but they also allow students to practice communication skills by articulating their feedback verbally reviews (reviews are highly interactive between presenters and reviewers) and in writing, and they provide the students with an opportunity to practice receiving feedback gracefully. During the activity, the instructors observe the peer reviews, and afterwards briefly review the written feedback provided by the students. Our observation during these activities was that students seemed to find the feedback useful. However, when we assessed the students' ability to perform a formal review of a presentation or a visualization individually, as discussed below, we found the feedback was often not specific or actionable. *In future course offerings, we plan to instruct students on how to provide specific and actionable feedback prior to the informal reviews and to update our informal peer review worksheet to determine the extent to which the feedback they have written is actionable and specific.*

Formal Reviews. In order to assess the students' ability to provide formal peer feedback, we created a set of rubrics for the students to assess an oral presentation and the visualizations in

these presentations. Following a presentation, the students had 5-8 minutes to write their assessments. The rubrics were published on the course website ahead of time. We also reviewed the rubrics together as a class. The students used the rubrics on three graded assignments, each of which is an individual assignment (no collaboration is permitted). In the first capstone assignment, 12% of the grade is based on the students' ability to assess and provide constructive feedback on their own and on other students' in-class project presentations using the rubrics. When grading the first capstone assignment feedback, we noticed that students frequently failed to give specific and actionable feedback. We subsequently provided JiTT instruction to teach the students how to provide feedback using the rubrics. We also provided examples of "good" feedback, so that in the second capstone assignment, students had instructor feedback from the first assignment along with the JiTT instruction to prepare them for the second round of presentations and for the final exam. We also learned from discussions with the students after the first capstone assignment that the rubrics were too long and too complicated to use effectively during a presentation (i.e., it was difficult to follow a presentation and observe all of the items in the rubric; it was also difficult in the 5-8 minutes to process and write the assessment). To address these issues, we tried to reduce the number of rubrics used by the students in their second capstone assignment and in the final exam. We were able to eliminate and consolidate the rubrics, going from ten to six. Unfortunately, the number increased to nine to account for new rubrics related to presentation delivery (e.g., blocking and gestures) in the second capstone assignment. The student rubrics are shown in [Appendix D](#). A more extensive set of rubrics was used by the instructors to assess student performance. These rubrics are shown in [Appendix E](#).

In the second capstone assignment, 6% of the students' grade is based on their ability to use the updated rubrics to assess their peers' (and their own) presentations and visualizations. In the take home final, 48% of the final exam grade is based on the students' ability to assess two oral presentations by students in the year-long capstone course, including the visualizations contained in those presentations, using the rubrics provided. On the final exam, the instructions also specify the feedback should account for significance (i.e., the comment addresses at least one aspect of the talk that affects the audience's ability to understand a main takeaway), and justify the feedback's significance (i.e., the comment explains why the audience's ability to understand a main takeaway is affected), both of which were necessary in order to receive full credit for the feedback.

We have not yet had a chance to fully analyze the effectiveness of our teaching methods or assessments related to this objective. Based on our observations during the in-class activities, students were able to provide useful feedback to their peers; we presume it was specific and actionable—at least to some degree. However, when the assignments required the students to provide formal feedback using the rubrics, students tended to simply repeat the words in the rubrics, rather than provide specific details about the presentation or visualization. For instance, one student recorded feedback regarding visualization usage in the first capstone presentation, "All diagrams present and explained" rather than describing how the explanations enhanced the presentation. Another student commented on the slide format "Sometimes I felt as if there was too much on the slides—both diagrams were a bit overwhelming" rather than provide actionable feedback or feedback on particular slides that exhibited problems. After grading the final exam, it appears that at least some of the students were better able to use the updated rubrics provided by the instructors to write specific, actionable feedback (both positive and negative) at the end of

the semester. For instance, one student provided the following feedback on visualization quality “The visualizations are decent overall, however, the flowchart failed to convey hierarchy, grouping, and sequence. At first glance, I didn’t know where to start looking...To achieve this they could make a clear starting place and have shown grouping and/or hierarchy.” And another student provided feedback on the level of detail and use of terminology with the following comment, “The presenters did a good job of explaining terms that were necessary for the understanding of the project. Terms like pull were explained at a level that was acceptable to the audience. In the future, including a visualization of the pull process would reduce the amount of time explaining the term.” Both of these comments have more of the attributes of the feedback we expected.

Although the SOFT 261 students seem able to provide informal feedback during guided exercises, they struggle with providing formal feedback. They did a good job of providing both positive and negative feedback, and in providing constructive feedback, but they struggle with providing specific, actionable and significant feedback. They also struggle to justify how the suggested changes can help improve the artifact. *In future course offerings, we plan to provide instruction for writing good (specific, actionable, significant) feedback, explain how the investment in writing good feedback can pay off for both the reviewers and the presenters, and illustrate how good feedback is specified. We also need to consider giving the students more time to process the presentation and to write their feedback (and to make sure it exhibits all of the criterial we have specified).*

Learning Objective 3

Assessing the [third learning objective](#) in SOFT 261 is more challenging. Through the capstone project assignment students applied disciplined software engineering principles and practices by completing a software construction project (Phase I of the capstone assignment) and a software maintenance project (Phase II of the capstone assignment). Both assignments were performed in teams of three to four students using real-world software. The extent to which students met this learning objective can be assessed based on our observations of the student sduring lab and during class, and based on their project status and plans recorded in the project management tool and their messages in Slack. During the weekly labs, students demonstrated their application of the Agile process to the teaching assistants through the various lab activities. Most students attended all of the lab sessions (attendance is required) and completed all of the checkpoints. Students also performed weekly stand-up meetings in class for the instructors to observe. Teaching assistants and instructors also monitored students’ Slack channels, team repositories, and project management artifacts to assess the students’ use of Agile practices. Based on these observations, the students appeared to meet course objective 3.

We can also measure student learning through the students’ capstone assignment grades. On the first assignment student scores ranged from 72% to 88%. Student teams performed well building a module using the OpenMRS framework, applying the Agile process methodology, and on demonstrating visual communication techniques. All but one team earned all six points on the application of Agile processes (the other team earned five points). Three of the five teams earned full credit for demonstrating visual communication techniques; the other two teams scored a five out of six. Where the students did not do well was on software engineering practices related to

testing, documentation, and practices to support software maintainability—all of the practices they had learned and used in previous core course projects. Surprisingly, most teams lost the majority of their assignment points in this part of the assignment. The highest number of points earned in this category was three out of six points (the overall assignment was worth 33 points). Two out of five teams earned three points, two teams earned two points, and one team earned only one point for software engineering practices. We believe that the students’ poor performance on software engineering practices was at least partially due to being overwhelmed with the independent learning and the less structured assignments in the course, and that they treated these tasks as lower priority when they fell behind on the assignment.

On the second capstone assignments, student scores ranged from 85% to 98%. Table 1, shown below, shows the number of points earned by each team in each category for the team component of the assignment (10 additional points were awarded based on the individual’s performance). Most teams lost a point in the application of Agile processes for not writing user stories from the perspective of the user. User stories were a difficult concept for students to learn and we learned that we need to provide more instruction on how to identify and specify user stories. Most teams effectively demonstrated visual communication skills in their capstone presentation. Students lost points for a variety of reasons, including failing to include a required visualization and professional writing in the presentation slides.

	Possible Points	Team A	Team B	Team C	Team D	Team E
Contributions to OpenMRS	24	22	24	24	24	24
Professional Communication Practices	8	8	8	8	8	8
Application of Agile Process methodology and tools	8	7	7	7	7	7
Demonstration of Visual Communication Techniques	15	11.5	13	12.5	14.5	12.5
Total Team Score	55	48.5	52	51.5	53.5	51.5

Table 1. SOFT 261 Capstone Phase II Team Scores

Finally, student success with respect to this objective can also be assessed by the number of OpenMRS talk threads the students participated in (19), the number of JIRA tickets the students commented on or worked on (21), and the number of pull requests each team worked on (21). These numbers, though raw with no baseline for comparison, show that the student teams were actively (and successfully) working on the OpenMRS project and interacting with the project developers. *For future course offerings, we plan to explore ways to better assess student achievement of this outcome, including ways to leverage the data collected during the inaugural course offering to compare with future course offerings.*

Analysis of Student Perceptions

Software engineering students progress through the program as a cohort, and they have thus far had the same set of instructors for the core courses. This consistency in the student population and their shared academic history have enabled us to develop a course that builds on the themes set in the previous courses and to also leverage our knowledge of the students’ backgrounds and capabilities. It has also presented an unexpected challenge in that students expect the course to be

very similar to the previous core courses in structure and teaching methodologies. When we made structural changes to SOFT 261 (e.g., removed some of the supporting framework provided in the previous three core courses and incorporated more independent learning activities), the students expressed concern and frustration at the beginning of the semester. Despite these changes, however, students became more confident in their ability to work without all of the scaffolding as the semester progressed. One student even commented “While the project was intimidating at first, it ended up being very helpful.”

To analyze students’ perceived learning and attitudes towards the course, we developed a brief survey that was administered in the 3rd, 8th, and 16th weeks of class. The survey statements are shown below in Table 2. The survey also included space for comments. The survey was administered on paper during class. Surveys were collected by a student in the course and placed in an envelope that was delivered to the instructor at the end of class. Based on the number of responses, participation on all three surveys was 100%.

#	Statement	Strongly Disagree (1) Strongly Agree (5)				
		1	2	3	4	5
1	The amount of course work is reasonable.					
2	The homework and journal assignments help me understand and apply the subject matter.					
3	The lab assignments help me understand and apply the subject matter.					
4	The in-class research activities help me understand and apply the subject matter.					
5	The in-class peer instruction activities help me understand and apply the subject matter.					
6	The course project helps me understand and apply the subject matter.					
7	The format of the labs provides enough guidance to complete the lab.					
8	Communication skills are an important topic for software engineering students to study.					
9	I prefer to study communication skills in a software engineering course.					
10	I feel more confident producing and delivering visual communications related to software architecture, implementation, planning and tracking.					
11	I feel more confident formulating constructive feedback on visual communications.					
12	I feel more confident working in a team to communicate technical information.					

Table 2. SOFT 261 Student Survey Questions

Figure 1, shown below, displays the aggregated data across the three surveys. For each survey statement shown on the x-axis (S1—S12), the mean of the students’ scores is shown on the y-axis. The scores are based on a Likert scale of 1 to 5 where a score of 1 indicates the student “strongly disagrees” with the survey statement and a score of 5 indicates the student “strongly agrees” with the survey statement. With the exception of S2 (“The homework and journal assignments help me understand and apply the subject matter.”), student agreement with the

survey statements increased over the semester. Based on comments provided in the survey responses, we believe the reason for the drop in agreement with S2 is that students did not perceive value in the journal assignments. For instance, one student commented “I liked doing journal assignments in the earlier software engineering courses but now they are starting to feel like a waste of time especially with other classes having large projects...,” and another student stated “The homework is helpful but the midterm took too much time ... Journals feel unnecessary and just add to the stress.” We also found that many students did not complete the journal assignments despite the fact that the journal grades account for 5% of the students’ final grade and were graded only for effort and professional writing (versus correctness).

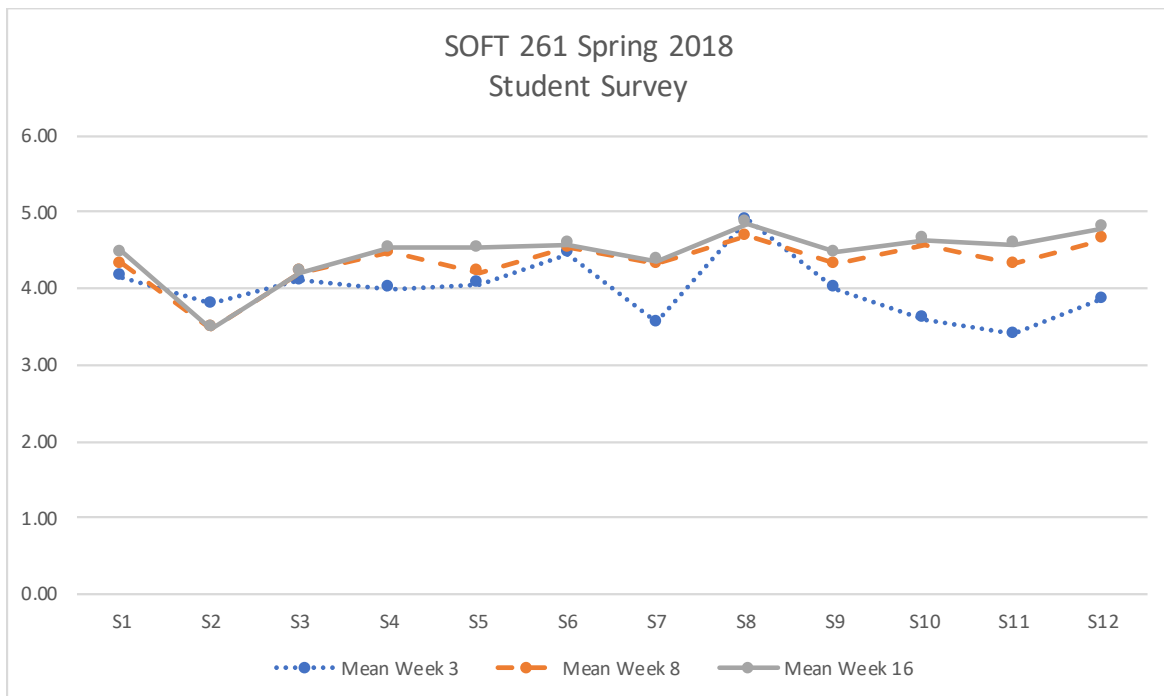


Figure 1. SOFT 261 Student Survey Results

Although we believe that the differences between SOFT 261 and the previous core courses in terms of infrastructure and independent learning will continue to be an issue in future course offerings, we plan to mitigate some of the discomfort for students by explaining the reasons for the changes at the beginning of the semester, and by scaling back on some of the independent learning activities until later in the semester.

Summary of Planned Changes

While we were overall very satisfied with the course as it was taught during its inaugural offering, we plan to continue to evolve the course to improve student learning and to improve the scalability of the course (since we anticipate having twice as many students in the course when it is offered again next year).

The first set of changes is related to providing additional course materials. We noticed that students can achieve some level of self-learning during the fourth semester, but it is more limited than we expected. Also, we noticed that many students do not take notes during class. We are not sure if they believe what we are teaching is common knowledge or if they were expecting the course text book we developed for the other core software engineering courses to be updated and available for reference. Given the unique combination of topics presented in this course and the areas we observed students struggling, we plan to develop course materials covering the following topics for the a course:

- Software process models
- Specifying requirements (e.g., user stories)
- Task estimation and planning
- Risk identification, assessment and mitigation
- How to create a visual communication
- How to write an agenda
- Learning software architecture (e.g., MVC)
- Software development workflow (e.g., Jira and Github)
- How to find open tasks in an open source project (e.g., OpenMRS)
- How to give specific, actionable constructive feedback

The course materials will continue to include in-class worksheets similar to the worksheets designed for the inaugural offering of the course. These worksheets will be used to give students hands-on practice working in their teams during class and then assigned as homework if not completed in class. We will also convert our lecture notes into an on-line text book that covers the instructional material delivered in class. The course material will also include model examples of “good” and “weak” artifacts (e.g., user stories, peer feedback).

Designing a course that teaches visual communications in the context of software engineering was a challenging endeavor. Not only did we not have experience teaching this novel combination of topics, but we also were unsure how to assess student learning. We were also in a situation of deploying the fourth new course in four semesters, and as a result, entered the semester with limited preparation. While we believe our assessments were adequate, we also believe they require significant improvements. The second major change planned for the next instantiation of SOFT 261 is a redesign of the course assessments. Our preliminary list of ideas includes:

- Provide more formative assessments earlier in the semester. For instance, the take home midterm came after spring break. The feedback on the midterm was then almost too late to help the students with the take home final,
- Ensure that the journal assignments are integrated with the rest of the course,
- Decide if quizzes over concepts are necessary, and if so, develop a set of quizzes that can be easily graded as the cohort size grows,
- Create assessments that can be graded in a timely manner as the cohort size grows, and
- Assess student peer feedback earlier in the semester.

Although we expect these planned changes to have a positive impact on student learning, there are other aspects of the course that we do not yet know how to change in order to improve student learning and performance. In particular,

- How to motivate students to use the assignment grading breakdown as a checklist to make sure they are submitting a complete assignment. The assignments have multiple steps and components. For each part of an assignment, we list the number of points that are possible; however, students often turn in incomplete work.
- How to motivate students to use professional writing in all of their submissions (e.g., correct punctuation, spelling, grammar, etc.).
- How to explain the value and importance of reflective assignments.

Summary and Overall Assessment of the PRT Portfolio Process

Preparing a benchmark portfolio was beneficial in several ways. First, the PRT portfolio process provides structure and guidance in how to design (or re-design) a course. It also provides a community of faculty from across UNL whom I can learn from and with whom I can share my teaching experiences. After working through this process and maintaining a course reflections journal while teaching this course, I am much better prepared to create a course and I am much more confident in the effectiveness of a course developed using this process. Furthermore, I am confident in what I have learned to the extent that I can share my experiences with other faculty members, and have already begun to do so with a new faculty member in our department. Through the development of this portfolio I learned how to avoid the trap of letting a textbook table of contents drive the organization of a course. Instead, I begin by writing a reasonable number of measurable course objectives, and then develop course activities and materials to support those objectives, and design assessments to measure student learning of the course objectives. While this is a seemingly simple process, there are many challenges, and much more intellectual effort is required. Writing the final course portfolio paper was also a useful exercise in assessing the effectiveness of the teaching methods, course materials, outside activities and assessments. After reviewing each component, I was able to identify a set of changes that I believe will improve the course and that I can assess next time the course is offered.

The resulting portfolio has the potential to be a valuable resource to those who review my professional development, to those who are interested in developing a course that teaches communication skills in the context of software engineering, and to future instructors of the course.

Appendix A

Course Syllabus & Schedule

SOFT 261 Syllabus Spring 2018

Prerequisites

- A grade of C+ or higher in SOFT 260.

Meeting Times

- Classes: 11:00-12:15 TR
- Labs: 8:30-10:20 F

Instructor(s)

- Suzette Person — 362 Avery Hall (sperson@cse.unl.edu)
 - Office Hours: By appointment
- Brady Garvin — 356 Avery Hall (bgarvin@cse.unl.edu)
 - Office Hours: By appointment

Teaching Assistants

- Sara El Alaoui (GTA) — 12 Avery Hall (ea.sara@ymail.com)
 - Office Hours: Posted on Piazza
- Jim Drake (UTA) — 12 Avery Hall (jimdrake55x@gmail.com)
 - Office Hours: Posted on Piazza

Textbook

- No assigned textbook

Course Description

From the official course description:

Techniques and tools based on disciplined software engineering principles for producing, interpreting, and communicating visual artifacts related to software architecture and construction; techniques for communicating with technical and non-technical audiences. Techniques for managing software projects, communicating and collaborating effectively in teams, and visualizing software process models.

Course Objectives

After completing this course, students should be able to:

1. Produce and deliver visual communications related to software architecture, software implementation, and software planning and tracking to technical and non-technical audiences.
2. Formulate and communicate constructive feedback on visualizations and content in peer technical communications.
3. Work effectively in teams to achieve project and team goals, communicate technical information and to resolve conflicts.

Course Topics and Tentative Schedule

A detailed course schedule is available on the [course website](#).

Communication

Communication and announcements from the instructor(s) will be via the course Piazza page at <<https://piazza.com/unl/Spring2018/soft261>> or in rare cases via email. It is CSE Department policy that all students in CSE courses are expected to regularly check their email so they do not miss important announcements.

The primary medium for contacting the instructor(s) or TA(s) is the course Piazza page. Questions about course content or questions that are of general interest to other students should be posted there.

The instructor(s) and teaching assistant(s) also have regular office hours. They may also be available by appointment (as their schedules permit); please schedule an appointment via email if your question is urgent or you cannot attend regular office hours.

Additionally, the CSE Student Resource Center (SRC) in Avery 12 is staffed by student tutors who are available to help you with this course or with issues such as problems logging in to CSE systems, problems printing, printing installing an application, etc. The SRC also provides a study space that is open to all software engineering majors. The SRC website is [here](#).

The Department of Computer Science and Engineering also maintains an [anonymous suggestion box](#) that you may use to voice your concerns about any problems in the course or department if you do not wish to be identified.

Grading

Final grades will be based on:

Class participation	5%
In-class activities and project	40%
Final presentation and paper	15%
Quizzes	10%
Mid-term exam	10%
Homework assignments	15%
Journal assignments	5%

Letter grades will be assigned according to the following rubric:

- * A: 93–100, A-: 90–92
- * B+: 87–89, B: 83–86, B-: 80–82
- * C+: 77–79, C: 73–76, C-: 70–72
- * D+: 67–69, D: 63–66, D-: 60–62
- * F: 0–59

The instructor(s) will make every effort to grade and return submitted material within one academic week after the due date. If you have questions about your grade or believe that points were deducted unfairly, you must address the issue with one of the instructor(s) within one week after the graded assignment is returned to you. We will make every attempt to assign grades consistently on each assignment; we can do this only if we grade everyone's work at the same time.

As an ACE 2 course, the instructors will evaluate students' visual communication assignments using the [ACE 2D rubric](#).

SOFT 261 Journals

Reflection and writing are key elements of learning. Your homework assignments in SOFT 261 include a series of journal assignments. These exercises are intended to (1) help you prepare for upcoming in-class assignments, (2) provide opportunities for you to reflect on your learning and experiences in the course, (3) provide opportunities for you to practice and improve your written communication skills, and, (4) to be another way for you to communicate with us (the instructor[s]). We will also use your journal entries to identify common misconceptions, and topics that may warrant more (or less) discussion in the future.

Each week, a subset of journals will be selected at random for review and grading. Journal entries will be scored for *effort* (0 points or 1 point) and *professional writing* (0 points or 1 point). They are not scored based on the correctness of the response; rather, they serve as a way for students to practice asking questions when they are unsure of an answer. The instructor(s) will do their best to respond to questions asked in the journals that are graded.

Exams and Homework

In general, there will be no make-up exams. Exceptions may be made in emergency situations. Documentation may be required.

ACE Compliance

This course fulfills the three credit hours of ACE Student Learning Outcome #2:
Demonstrate competence in communication skills in one or more of the following ways:

- a. by making oral presentations with supporting materials,
- b. by leading and participating in problem-solving teams,
- c. by employing communication skills for developing and maintaining professional and personal relationships, and
- d. by producing and/or interpreting visual information.

This course is primarily focused on ACE SLO #2d.

SOFT 261H introduces tools and techniques based on disciplined software engineering principles for producing, interpreting, and communicating visual artifacts related to software architecture and construction. This course covers techniques for effective communication of software architecture design, software complexity, software process models, and software plans and status to diverse audiences. This course offers numerous learning opportunities via interactive lectures, hands-on class activities, lab work, homework assignments, a course capstone project, and guest speakers. Students receive extensive hands-on opportunities to produce, interpret, critique, and refine visualizations for

technical and non-technical audiences. Peer-to-peer reviews of visualizations for adherence to visual communication principles, legibility, understandability, correctness, completeness, inconsistencies, etc. allow students to practice and learn from real-world review processes in addition to receiving instructor feedback and grade.

Traditional exams and quizzes will be utilized to assess content knowledge acquisition. The student's ability to effectively produce, interpret, critique, and refine visualizations for technical and non-technical audiences will be assessed using individual and team assignments and presentations. To demonstrate and practice the entire semester's content, students will complete a capstone project to assess their grasp of the concepts and their ability to effectively apply the tools and techniques. Students' visualizations will be assessed by the degree to which they articulate the features of the architecture design, complexity of the software, and program plans, and status documentation. Students will also be assessed on their ability to interpret and critique visualizations using criteria such as correctness, completeness, inconsistencies, etc. and their ability to effectively communicate constructive feedback for improving visualizations to better communicate the concepts and ideas contained therein. Student work will also be evaluated and assessed using the [ACE 2d rubric](#).

Computer Policy

The computer policy for this course is the same as the computer policy for the software engineering major, which is posted [here](#).

Technology Policy

Research has shown that digital distractions can have a negative impact on your grade and can be distracting to those seated near you. For these reasons, the use of cell phones, including texting, posting to social media, etc. is not permitted during class time under any circumstances. Leave your cell phone in your backpack during class time.

You are expected to bring your laptop to class every day. Ensure your battery is sufficiently charged in the event there is not an accessible power supply where you are sitting. Laptops may be used during class time for the purpose of taking notes and for in-class assignments only.

Collaboration Policy

In practice, software engineers work as part of a team. Therefore, in this course we will require you to work together to understand course concepts and assignments, and to practice working in teams. However, outside of your assigned groups, you may not develop joint solutions, share work, or copy anything. You are also responsible for safeguarding your own work. All external contributions must be acknowledged, including help from others or from non-course materials such as websites. If in doubt, ask.

Dead Week Policy

In compliance with UNL's 15th Week Policy (see the main Registration and Records [webpage](#)), be aware that the final assignment (project paper) will be due during the final week of classes. Further, there will be in-class assignments and presentations during the final week of class. Note also that all assignments, homework, labs, etc., will have a strict final due date during the final week of classes.

Academic Integrity

The Computer Science and Engineering department has an [[Academic Integrity Policy](#)], which all students enrolled in any software engineering course are bound by. You are expected to read, understand, and follow this policy. Violations will be dealt with on a case-by-case basis and may result in a failing assignment or a failing grade for the course itself.

Sources for Help and Assistance

You are ultimately responsible for your success in this course. If you have questions on material covered or assigned in class, it is up to you to seek out assistance from the course instructor(s) or TA(s). Staff in the [CSE Student Resource Center](#) may also be able to assist you with general questions. The CSE Department also maintains a [Frequently Asked Questions](#) page.

Accommodations

Students with disabilities are encouraged to contact an instructor for a confidential discussion of their individual needs for academic accommodation. This includes students with mental health disabilities like depression and anxiety. It is the policy of the University of Nebraska-Lincoln to provide individualized accommodations to students with documented disabilities that may affect their ability to fully participate in course activities or to meet course requirements. To receive accommodation services, students must be registered with the Services for Students with Disabilities (SSD) office, 232 Canfield Administration, 472-3787.

Course Schedule

Module I: Course Introduction		
Session	Learning Goals	Assignments
1.1	<ol style="list-style-type: none"> 1. Locate course objectives, roadmap, and resources. 2. Describe the components of effective communication. 3. Describe at least three challenges specific to communication in software engineering. 	<ul style="list-style-type: none"> - Read AI policy & course syllabus. - Sign-up on course Piazza site - Complete journal assignment - Set-up Git homework repo - Listen to Talking to Stakeholders: 13 Communication Anti-patterns that Block Good Ideas
1.2	<ol style="list-style-type: none"> 1. Describe communication anti-patterns that hinder effective communication. 2. Describe how communication is more than just sending and receiving a message. 3. Apply the main elements of visual communication. 4. Provide feedback on visual aspects of communication. 	<ul style="list-style-type: none"> - Read Elements of Visual Communication
1.3 (Lab)	<ol style="list-style-type: none"> 1. Identify, plan and assign tasks necessary to ramp up on a new software development project. 2. Coordinate research among team members to learn the tools and technologies needed to support work on a new software engineering project. 3. Setup a team communication tool. 	
1.4	<ol style="list-style-type: none"> 1. Coordinate research among team members to learn the basics of a software process model. 2. Identify the history, strengths, weaknesses, and application of the waterfall, "V", spiral, prototyping, and agile software process models. 3. Summarize and present the keys ideas of a software process model at the whiteboard. 	<ul style="list-style-type: none"> - Individual Homework 1.4 - Read Sec. I-IV(A) in What Makes a Great Software Engineer? - Complete journal assignment
1.5	<ol style="list-style-type: none"> 1. Prepare a proposal for a software project. 	<ul style="list-style-type: none"> - Read Agile Software Development

	<ol style="list-style-type: none"> Describe the major flavors of agile software processes. Use agile terminology appropriately. Write and derive user stories and tasks. 	(Sec. 1—5) - Read Scrum
1.6 (Lab)	<ol style="list-style-type: none"> Set up GitHub and Taiga to support team development of an OpenMRS module. Prepare a proposal for a software project. Populate a product backlog with epic(s) and stories. Plan a sprint, populating the sprint backlog from the product backlog. 	- Capstone Phase I assigned
Module 2: Capstone Phase I--Software Construction Project		
1/23	<ol style="list-style-type: none"> Explain how the agile methodology helps us manage problem complexity but not solution complexity. Explain how a good software architecture can help mitigate the effects of software change. Describe the breadth of change drivers that a software architecture should take into account. 	- Complete journal assignment
1/25	<ol style="list-style-type: none"> Explain the motivations for spending time estimating project tasks. Use planner poker to estimate sprint tasks. Use Taiga to record estimates and assign tasks. 	- Read Planning Poker
1/26 (Lab)	<ol style="list-style-type: none"> Plan a sprint, populating the sprint backlog from the product backlog. Estimate and assign tasks in the sprint backlog. Record your project and sprint plans in Taiga. 	
1/30	<ol style="list-style-type: none"> Assess the quality of a user story. Communicate status, plans and risks in a daily standup meeting. Communicate bad news in a professional manner. 	- Complete journal assignment - Read Section IV(B) in What Makes a Great Software Engineer?
2/1	<ol style="list-style-type: none"> Relate the process for visualizing communication to the process of developing software. Apply the process shown in class to a small scenario, transforming the ideas in the scenario into a visualization. 	
2/2 (Lab)	<ol style="list-style-type: none"> Demo a working version of your OpenMRS module for the TAs. Write high-quality user stories, incorporating feedback from the instructors and TAs. Plan a sprint, populating the sprint backlog from the product backlog. Record your project and sprint plans in Taiga. 	Quiz 2.6
2/6	<ol style="list-style-type: none"> Identify the motivations for documenting software architecture. Identify the main elements in a software architecture diagram. Apply the process for creating visualizations shown in class to begin documenting the architecture of your team's OpenMRS module. 	- Complete journal assignment - Read INVEST in Good Stories and SMART Tasks - Read INVEST in User Stories
2/8	<ol style="list-style-type: none"> Trace the mapping of an OpenMRS module to the MVC architecture components. Locate the services provided by OpenMRS. Trace the code in the basic OpenMRS module. 	
2/9 (Lab)	<ol style="list-style-type: none"> Demo a working version of your OpenMRS module for the TAs. Write high-quality user stories, incorporating 	Quiz 2.9

	<p>feedback from the instructors and TAs.</p> <ol style="list-style-type: none"> Plan a sprint, populating the sprint backlog from the product backlog. Record your project and sprint plans in Taiga. 	
2/13	<ol style="list-style-type: none"> Apply the process shown in class for creating a visualization. Create a visualization mapping your OpenMRS module to the MVC architecture. Formulate a set of scenarios that can help a reviewer analyze your diagram. 	<ul style="list-style-type: none"> - Complete journal assignment - Read Section IV(C) in What Makes a Great Software Engineer?
2/15	<ol style="list-style-type: none"> Apply the process shown in class for creating a visualization. Solicit useful, actionable feedback on a visualization in the context of a review. Provide useful, actionable feedback on a visualization in the context of a review. Take feedback professionally and graciously. 	
2/16 (Lab)	<ol style="list-style-type: none"> Demo a working version of your OpenMRS module for the TAs. Write high-quality user stories, incorporating feedback from the instructors and TAs. Plan a sprint, populating the sprint backlog from the product backlog. Record your project and sprint plans in Taiga. 	Quiz 2.12
2/20	<ol style="list-style-type: none"> Design a project handoff presentation. Develop a clear, concise presentation that incorporates appropriate visualizations to describe your OpenMRS module. Plan the presentation delivery in a way that balances the participation among team members. 	<ul style="list-style-type: none"> - Complete journal assignment - Read Section IV(D) in What Makes a Great Software Engineer? - Read Storytelling-The Missing Art in Engineering Presentations
2/22	<ol style="list-style-type: none"> Apply the process shown in class for creating a project handoff presentation. Solicit useful, actionable feedback on a presentation in the context of a review. Provide useful, actionable feedback on a presentation in the context of a review. Take feedback professionally and graciously. 	
2/23 (Lab)	<ol style="list-style-type: none"> Close out a project in Taiga. Perform a project retrospective. Create a module-evolution retrospective diagram. 	- Capstone Phase I due today
2/27	<ol style="list-style-type: none"> Work as a team to deliver a project handoff presentation. Use visualizations in a presentation to communication software architecture and software evolution. Provide a constructive qualitative assessment of a project handoff presentation. 	<ul style="list-style-type: none"> - Complete journal assignment - Complete 360 review
3/1	<ol style="list-style-type: none"> Work as a team to deliver a project handoff presentation. Use visualizations in a presentation to communication software architecture and software evolution. Provide a constructive qualitative assessment of a project handoff presentation. Setup a Scrumban project in Taiga. 	
Module 3: Capstone Phase II--Software Maintenance Project		
3/2	<ol style="list-style-type: none"> Set up Taiga and Slack to support team maintenance 	- Capstone Phase II assigned

(Lab)	<ul style="list-style-type: none"> 2. Identify subprojects within a large codebase that your team can contribute to. 3. Identify reasonably scoped and useful maintenance tasks to begin working on. 	
3/6	<ul style="list-style-type: none"> 1. Work as a team to perform maintenance tasks on OpenMRS. 2. Communicate status, plans and risks in a daily standup meeting. 	- Complete journal assignment
3/8	<ul style="list-style-type: none"> 1. Research and ramp-up on a software development process. 2. Create a pull request. 3. Work as a team to perform maintenance tasks on OpenMRS. 	
3/9	<ul style="list-style-type: none"> 1. Assess and record project progress, 2. Demo a working version of your OpenMRS contributions (if your team is at the end of a sprint), 3. Make course corrections as necessary, 4. Perform software maintenance on unfamiliar code, and 5. Communicate project status to someone outside your team. 	
3/13	<ul style="list-style-type: none"> 1. Create appropriate visualizations in technical documentation or create appropriate visualizations to represent documentation changes. 2. Create appropriate visualizations to represent the impact of testing changes. 3. Create appropriate visualizations to represent code changes related to a bug fix or feature enhancement. 	
3/15	<ul style="list-style-type: none"> 1. Plan a status meeting. 2. Prepare a meeting agenda. 3. Draft an email message to send with the agenda. 4. Create a slide template that can be used for status meetings. 	- Homework 3.6 assigned
3/16 (Lab)	<ul style="list-style-type: none"> 1. Assess and record project progress, 2. Demo a working version of your OpenMRS contributions (if your team is at the end of a sprint), 3. Make course corrections as necessary, 4. Perform software maintenance on unfamiliar code, and 5. Communicate project status to someone outside your team. 	
3/20	SPRING BREAK	
3/22	SPRING BREAK	
3/23	SPRING BREAK	
3/27	<ul style="list-style-type: none"> 1. Explain why drawing is not art. 2. Explain why drawing is a useful communication practice for software engineers. 3. Describe the basic tools needed to visually communicate in software engineering. 	<ul style="list-style-type: none"> - Complete journal assignment - Take home midterm assigned - Homework 3.8 assigned
3/29	<ul style="list-style-type: none"> 1. Lead a status meeting with the project stakeholders. 2. Take meeting minutes. 3. Perform software maintenance on unfamiliar code. 	
3/30 (Lab)	<ul style="list-style-type: none"> 1. Assess project progress. 2. Make course corrections as necessary. 	

	3. Perform software maintenance on unfamiliar code.	
4/3	1. Explain the challenges of testing highly configurable software. 2. Explain how and when combinatorial interaction testing is used in software testing.	- Complete journal assignment
4/5	1. Lead a status meeting with the project stakeholders. 2. Take meeting minutes. 3. Perform software maintenance on unfamiliar code.	
4/6 (Lab)	1. Assess and record project progress. 2. Make course corrections as necessary. 3. Plan the visualizations for your final presentation. 4. Perform software maintenance on unfamiliar code.	
4/10	1. Design a presentation for a release meeting. 2. Pre-plan important aspects of a presentation's delivery, including blocking, gestures, tempo, and team coordination.	- Complete journal assignment
4/12	1. Lead a status meeting with the project stakeholders. 2. Take meeting minutes. 3. Perform software maintenance on unfamiliar code.	- Take home midterm due today
4/13 (Lab)	1. Assess and record project progress. 2. Make course corrections as necessary. 3. Plan the visualizations for your final presentation. 4. Perform software maintenance on unfamiliar code.	
4/17	1. Work as a team to prepare a release meeting presentation. 2. Work as a team to finalize a software maintenance project.	
4/19	1. Solicit useful, actionable feedback on a presentation in the context of a release meeting. 2. Provide useful, actionable feedback on a presentation in the context of a release meeting. 3. Take feedback professionally and graciously.	- Homework 3.8 due today
4/20 (Lab)	1. Close out your OpenMRS maintenance project. 2. Perform a project retrospective. 3. Finalize your release meeting presentation.	
4/24	1. Present your team's capstone project. 2. Provide a constructive qualitative assessment of a release meeting presentation.	- Complete 360 review
4/26	1. Present your team's capstone project. 2. Provide a constructive qualitative assessment of a release meeting presentation.	
4/27 (Lab)	1. Critically assess and formulate specific and actionable feedback on a formal presentation, 2. Create or improve a visualization to communicate key information clearly, creatively, and concisely. 3. Develop and express an effective argument of how the new or enhanced visualization would improve the presentation.	- Take home final exam assigned
5/1	Final Exam	- Take home final exam due 5:30

Appendix B

Homework 4.1 versus Homework 3.8 – Features Matrix Assignment

Student A Features Matrix **Homework 1.4**

	Waterfall	Non-Agile Iterative	Agile
Adaptable		X	X
Client Focused		X	X
Concrete Steps	X		
Group Oriented		X	X
Iterative		X	X
Lengthy	X		
Releasable in One Cycle	X		X
Well Documented	X		

Student A Features Matrix **Homework 3.8**

	Scrum	Scrumban	Kanban
Daily Standup Meetings	Yes	Maybe	No
Develops in Short Sprints	Yes	Maybe	No
Each Participant Has a Distinct Role	Yes	Maybe	No
Organized Around Small Teams	Yes	Yes	No
Assigns Tasks To User Stories	Yes	Maybe	No
Limits Work In Progress	No	Maybe	Yes
Limits Ready Work	No	Maybe	Yes
Taskboard Can Span Multiple Teams	No	Maybe	Yes

Student B Features Matrix Homework 1.4

	Waterfall	Non-agile iterative	Agile
Good for large projects			<input type="checkbox"/>
Structured in phases	<input type="checkbox"/>		
Sticks to original plan	<input type="checkbox"/>	<input type="checkbox"/>	
Flexible			<input type="checkbox"/>
Strong documentation	<input type="checkbox"/>		
Has stabilization phase		<input type="checkbox"/>	<input type="checkbox"/>
QA can be done during implementation		<input type="checkbox"/>	<input type="checkbox"/>
Product Owner determines scope	<input type="checkbox"/>	<input type="checkbox"/>	
Entire team responsible for work			<input type="checkbox"/>
Scheduled meetings	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Provides a final product	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Student B Features Matrix Homework 3.8

Feature \ Model	Scrum	Scrumban	Kanban
Works in Iterations (Sprint, etc.)	Yes	Maybe	No
Utilizes a Backlog for US/Tasks	Yes	Yes	Yes
Allows for Measurable Productivity	Yes	Maybe	No
Client Sets Priority	Yes	No	No
Continuous Workflow	No	Yes	Yes
Pre-defined Roles Per Team Member	Yes	No	No
Work-in-Progress Limits	No	Yes	Yes
On-Demand Planning	No	Yes	Yes
Allows for Highly Variable Environment	No	Yes	Yes

Student C Directed Paraphrasing **Homework 1.4**

New employee,

Welcome to the team! You should settle in well since you are more than qualified for this position. I understand that your education was mainly focused on the agile software development, and I am writing this to assist in the understanding of how we operate. As a software company focused on developing small projects, we initially found that using the waterfall methodology was the ideal process model to use.

It benefits us by being able to strongly document our software before we implement it. This will be beneficial to you as a newcomer, so you are not lost in the project we are currently focused on. Along with that, we are able to have a clearly set phases that will aid in our software development. Having a predetermined finished product can be good in some cases, similar to ours.

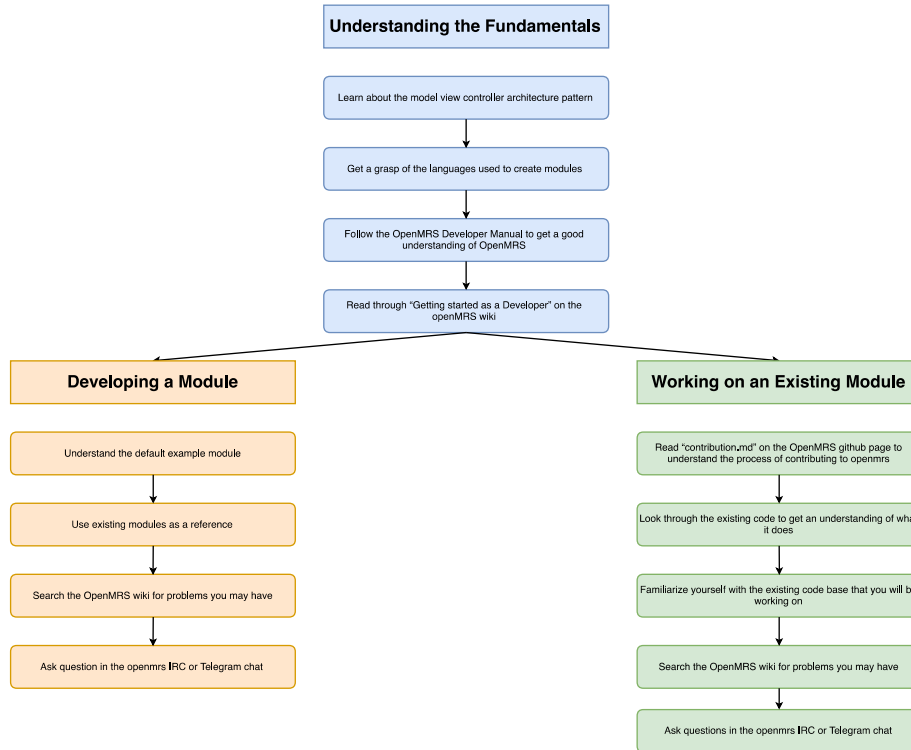
It should not be too difficult to adapt to this model. I understand having little practice in a new area is difficult at first, but over time it should become second nature, much like your working with the agile methodology. We will not be testing during implementation, so you will most likely have to revisit code should a quality assurance employee spot something. Another main difference is the length of our development, we will not be working in sprints, but rather in phases. This goes from project planning to implementation to testing and release.

I have been thinking of what you may introduce to our team from the agile methodology. We might want to start testing during implementation to save company time. I am sure that you can come up with some different methods for us to use over time should we be able to adjust accordingly.

Appendix C

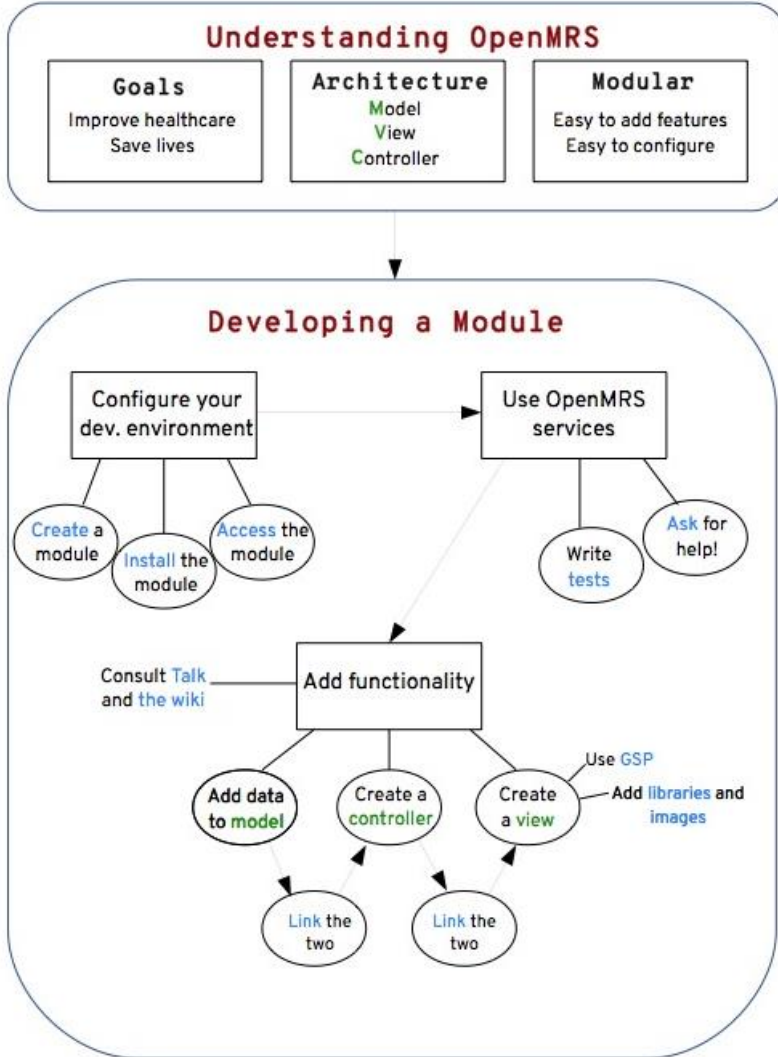
Take Home Midterm – Onboarding Process Visualization

Example of “C” level work visualizing an onboarding process

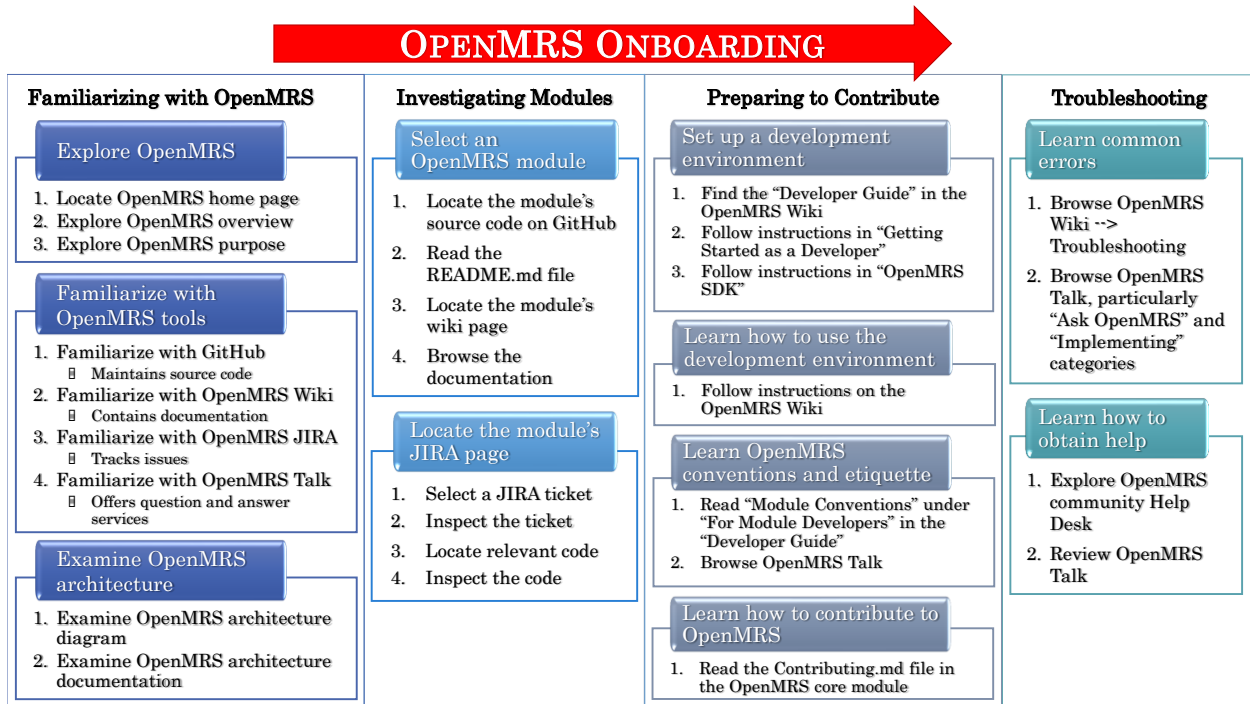


Example of “B” level work visualizing an onboarding process

OpenMRS Onboarding



Example of “A” level work visualizing an onboarding process



Appendix D

Presentation Rubrics—Student Version (after first capstone assignment)

	“A” level work	“B” level work	“C” level work	“D”/“F” level work
Slide Format	All slides use an unobtrusive theme, a readable font, and audience-friendly colors.	Most slides use an unobtrusive theme, a readable font, and audience-friendly colors.	Few slides use an unobtrusive theme, a readable font, and audience-friendly colors.	No slides use an unobtrusive theme, a readable font, and audience-friendly colors.
Visualization Quality	The visualizations are accurate and polished, and they effectively convey hierarchy, grouping and/or sequence.	The visualizations are accurate and polished, but do not effectively convey hierarchy, grouping and/or sequence.	The visualizations contain inaccuracies or are unpolished.	The visualizations are inaccurate and unpolished.
Visualization Usage	Visualizations are helpful and consistently well explained.	Visualizations are helpful and sometimes well explained.	Visualizations are unhelpful or not well explained.	Visualizations are unhelpful and not well explained.
Demo	Presentation includes a polished demo of the team’s contributions to OpenMRS and the team recovers gracefully from unexpected difficulties.	Presentation includes a unpolished demo of the team’s contributions to OpenMRS, or the team does not recover gracefully from unexpected difficulties.	Presentation includes an unpolished demo of the team’s contributions to OpenMRS, and the team does not recover gracefully from unexpected difficulties.	Presentation does not include a demo of the team’s contributions to OpenMRS.
Audience	Presentation is consistently appropriate for the audience in terms of level of detail and use of terminology.	Presentation is usually appropriate for the audience in terms of level of detail and use of terminology.	Presentation is sometimes appropriate for the audience in terms of level of detail or use of terminology.	Presentation is rarely or never appropriate for the audience in terms of level of detail or use of terminology.
Transitions	Transitions between topics are consistently smooth.	Transitions between topics are mostly smooth.	Transitions between topics are rarely smooth.	Transitions between topics are never smooth.
Blocking and Gestures	Speakers move deliberately, use effective gestures, and point at the screen as necessary; non-speakers show attention to the speaker or slides.	Speakers sometimes move deliberately, use effective gestures, and point at the screen; non-speakers show attention to the speaker or slides.	The blocking or gestures are distracting or absent, or speakers fail to point at the screen as necessary; non-speakers show attention to the speaker or slides.	The team’s blocking and gestures are consistently distracting or absent.
Tempo	Presentation pace is consistent, pauses are effective, and the audience is kept engaged.	Presentation pace is inconsistent, or needed pauses are missing, but the audience is kept engaged.	The presentation rushes or drags, and the audience occasionally becomes lost, bored, or disengaged.	The pace of the presentation consistently leaves the audience lost, bored, or disengaged.
Team Coordination	Presentation and question-answering responsibilities appear planned and team members coordinate professionally.	Presentation and question-answering responsibilities appear planned, and team members sometimes coordinate professionally.	Presentation and question-answering responsibilities appear planned, but team members do not coordinate professionally.	Presentation and question-answering responsibilities do not appear planned.

Appendix E

Presentation Rubrics—Instructor Version (after first capstone assignment)

Slides Rubric

	“A” level work	“B” level work	“C” level work	“D”/“F” level work
Content	Presentation includes a title slide, outline slides in the introduction and conclusion, and a final slide, and it transitions smoothly between topics.	Presentation is missing a title slide, outline slides in the introduction or conclusion, or a final slide, or it does not transition smoothly between topics.	Presentation is missing multiple structural slides, or it is missing one such slide and does not transition smoothly between topics.	Presentation is missing multiple structural slides and does not transition smoothly between topics.
Professional Writing	Presentation uses consistent, formal writing and is free of spelling and grammatical errors.	Presentation contains a few inconsistencies, informalities, spelling errors and/or grammatical errors, but they do not distract from the presentation.	Presentation contains inconsistencies, informalities, spelling errors and/or grammatical errors, and they sometimes distract from the presentation.	Presentation contains inconsistencies, informalities, spelling errors and/or grammatical errors, and they regularly distract from the presentation.
Slide Format	All slides use an unobtrusive theme, a readable font, and audience-friendly colors. Slide numbers or other indications of progress are included.	Most slides use an unobtrusive theme, a readable font, and audience-friendly colors. Slide numbers or other indications of progress are included.	Either few or no slides use an unobtrusive theme, a readable font, and audience-friendly colors, or else the slides lack a visual indication of progress.	Few or no slides use an unobtrusive theme, a readable font, and audience-friendly colors. The slides lack a visual indication of progress.
Visualization Usage	Presentation includes visualizations of all three contributions, and all visualizations are explained.	Presentation includes visualizations of all three contributions, but some of the visualizations are not explained.	Presentation is missing some of the required visualizations or none of the visualizations are explained.	Presentation does not include any of the required visualizations.
Visualization Quality	The visualizations are accurate and polished, and they effectively convey hierarchy, grouping and/or sequence.	The visualizations are accurate and polished, but do not effectively convey hierarchy, grouping and/or sequence.	The visualizations contain inaccuracies or are unpolished.	The visualizations are inaccurate and unpolished.

Presentation Content Rubric

	“A” level work	“B” level work	“C” level work	“D”/“F” level work
Audience	Presentation is consistently appropriate for the audience in terms of level of detail and use of terminology.	Presentation is usually appropriate for the audience in terms of level of detail and use of terminology.	Presentation is sometimes appropriate for the audience in terms of level of detail or use of terminology.	Presentation is rarely or never appropriate for the audience in terms of level of detail or use of terminology.
Balance	Presentation is balanced in terms of team participation, and all transitions between team members are smooth.	Presentation is balanced in terms of team participation, and some transitions between team members are smooth.	Presentation is not balanced in terms of team participation, or transitions between team members are not smooth.	Presentation is not balanced in terms of team participation, and transitions between team members are not smooth.
Use of Time	Presentation covers all important information without going short or long and leaves time for questions.	Presentation covers all important information but runs a little short or long.	Presentation covers only some important information or runs very short or very long.	Presentation covers no important information or covers only some important information while running very short or very long.
Demo	Presentation includes a polished demo of the team’s contributions to OpenMRS and the team recovers gracefully from unexpected difficulties.	Presentation includes a unpolished demo of the team’s contributions to OpenMRS, or the team does not recover gracefully from unexpected difficulties.	Presentation includes an unpolished demo of the team’s contributions to OpenMRS, and the team does not recover gracefully from unexpected difficulties.	Presentation does not include a demo of the team’s contributions to OpenMRS.
Backup Slides	Presentation includes several backup slides that are relevant to the types of questions that may be asked.	Presentation includes one backup slide that is relevant to the types of questions that may be asked.	Presentation includes backup slides that are not relevant to the types of questions that may be asked.	Presentation does not include backup slides to support Q&A.

Presentation Delivery Rubric

	“A” level work	“B” level work	“C” level work	“D”/“F” level work
Blocking and Gestures	Speakers move deliberately, use effective gestures, and point at the screen as necessary; non-speakers show attention to the speaker or slides.	Speakers sometimes move deliberately, use effective gestures, and point at the screen; non-speakers show attention to the speaker or slides.	The blocking or gestures are distracting or absent, or speakers fail to point at the screen as necessary; non-speakers show attention to the speaker or slides.	The team’s blocking and gestures are consistently distracting or absent.
Tempo	Presentation pace is consistent, pauses are effective, and the audience is kept engaged.	Presentation pace is inconsistent, or needed pauses are missing, but the audience is kept engaged.	The presentation rushes or drags, and the audience occasionally becomes lost, bored, or disengaged.	The pace of the presentation consistently leaves the audience lost, bored, or disengaged.
Team Coordination	Presentation and question-answering responsibilities appear planned and team members coordinate professionally.	Presentation and question-answering responsibilities appear planned, and team members sometimes coordinate professionally.	Presentation and question-answering responsibilities appear planned, but team members do not coordinate professionally.	Presentation and question-answering responsibilities do not appear planned.

Bibliography

1. *Software Engineering 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering, A Volume of the Computing Curricula Series*, Joint Task Force on Computing Curricula, IEEE Computer Society and Association for Computing Machinery, 23 February 2015.
2. *Understanding by Design*. Wiggins, Grant and McTighe Jay, 2005.
3. *The Responsive Classroom Discussions: The Inclusion of All Students*. Lyman, Frank, 1981. A. Anderson (Ed.), Mainstreaming Digest, College Park: University of Maryland Press, pp. 109-113.
4. *What Makes a Great Software Engineer*. Li, Paul Luo, Ko, Andrew J., Zhu, Jiamin. Proceedings of the 37th International Conference on Software Engineering. 2015, pp. 700-710.