Spring 2020 CQI Report
Software Curriculum

September 24, 2020

Committee: Omar Badreddin, Yoonsik Cheon, Oscar Mondragon, Daniel Mejia, and Salamah Salamah.

The Software cluster consists of four courses: CS 3195 Junior Professional Orientation, CS 3331 Advanced Object-Oriented Programming, CS 4310 Software Engineering I, and CS 4311 Software Engineering II. The committee and the course instructors met on September 18, 2020, and reviewed the assessments of the learning outcomes provided by the course instructors; see the appendix for the individual assessment reports prepared by the instructors.

- CS 3195 by Daniel Mejia
- CS 3331 by Omar Badreddin and Oscar Mondragon
- CS 4310 by Elsa Tai Ramirez
- CS 4311 by Steve Roach

The stated course outcomes of all the courses were met except for a few outcomes that were neither met nor covered mainly due to time constraints, i.e., one-week loss caused by the ongoing public health situation. In CS 4311, however, about 39 % (7/18) of the assessed outcomes were met marginally with the average earned points in the range of 70% - 74%.

Revision of Course Outcomes (Need Faculty Approval)
CS 3195 and CS 4310 instructors proposed several changes to the current course outcomes, and the committee reviewed/discuss them carefully. Below are the proposed changes that the committee approved.

CS 3195: Junior Professional Orientation

- Add a new level 2 outcome on technical writing.
  - ⇨ 2e. Prepare a written document expressing proper technical writing in professional settings, through email, written reports, and evaluation of products/services.

CS 4310: Software Engineering I

- Move the outcome 2f to CS 4311 and make it a level 1 outcome.
  2f. Relate the importance of professional societies.
  - ⇨ (CS 4311) 1d. Relate the importance of professional societies.
- Introduce a new level 3 outcome on interview report.
  - ⇨ 3i. Construct an interview report.

CS 3195 Junior Professional Orientation
This course provides an overview to the Computer Science profession with an emphasis on ethics and the local and global impact of computing on individuals, organizations, and society. In Spring 2020, it was taught by Daniel Mejia using the textbook, *A Gift of Fire: Social, Legal, and Ethical Issues for Computing Technology* (Sara Basse, 5th edition, Pearson, 2018). The committee recommends to keep up with the good work – meeting all the course outcomes satisfactory with a 97% student passing rate.

CS 3331 Advanced Object-Oriented Programming

This course provides in-depth exposure to the object-oriented programming paradigm. In Spring 2020, two sections were offered and taught by Omar Badreddin and Oscar Mondragon. In both sections, the required textbook was *Object-Oriented Software Development Using Java*, (Xiaoping Jia, 2nd edition, Addison Wesley, 2003). The committee recommends to:

- Monitor the failure rate (24% in one section).
- Cover API docs (outcome 3.d) that were not covered and thus not assessed in one section.
- Consider updating the old required textbook. One choice would be *Object-Oriented Analysis, Design and Implementation* (B. Dathan and S. Raminath, Springer, 2015; free e-book through UTEP library) complemented with other e-books such as *UML@Classroom* (M. Seidl, et. al, Springer, 2012) and Java textbooks.
- Remove the Class-Responsibility-Collaboration (CRC) cards approach introduced in one of the sections because it is treated in-depth in CS 4311; instead, devote more time to the required topics such as GUI (MVC), multithreading, and networking.
- Consider performing an informal study on the preparedness of entering students -- whether they have an understanding of OO concepts and are equipped with basic OO programming skills. It was mentioned by a past instructor that about 70% of entering students couldn't write OO code; they wrote procedure code in Java.
- Consider sharing course materials and others between the instructors of different sections not only for the CQI but also for the consistency of course offerings.
- Ask for continued support for instructional assistants and qualified teaching assistants.

CS 4310 Software Engineering I

This course is the first semester of a two-semester capstone course in which students work with a customer to capture and specify requirements for a real-world application. In Spring 2020, it was taught by Elsa Tai Ramirez. The course textbook was *Requirements Engineering* (Hull, E., Jackson, K., and Dick, J., 4th edition, Springer, 2017). The committee recommends to:

- Monitor the three outcomes that were not covered due to time constraints: 1c. process improvement, 2e. impact on computing, and 2f. professional society.
- Work with the CS 4311 instructor to move the outcome 2f (professional society) to CS 4311. It was suggested to include a discussion and self-reflection exercise on professional societies.
- Continue incorporating technologies such as Google Jamboard to facilitate and improve student learning.

CS 4311 Software Engineering II

This is the second semester of a two-semester capstone course in which students design and implement a real-world application specified in CS 4310. In Spring 2020, it was taught by Steve Roach. The committee recommends to:

- In the report, include a brief description of the course and the offering, e.g., textbook, as well as a quick summary of the findings including observations and recommendations.
- In the report, include how the concerns/recommendations from the last CQI were addressed. There are a few recurring recommendations (copied below from 2018 CQI):
    a. Develop a way to measure the outcome 2j (analysis of non-functional properties) that wasn't assessed.
    b. Monitor the outcomes that were met "marginally" with average scores 70-74%, i.e., outcomes 2b (detailed design) and 2c (architectural styles). This might be a concern, as it repeats (see the 2016 CQI report).

    c. Discuss with department faculty (especially those teaching the Fundamental courses) ways to improve students' skills in formulating and writing assertions in the first-order logic. Students continue to score low on writing pre and post conditions as part of the detailed design (2b).
- Pay attention to three outcomes that were not covered (3h. non-functional properties), not assessed (2e. testing), and not met (2a. architectural styles).
- Monitor seven outcomes that were met "partially" (average scores 70-74%) and do an outcomes analysis before the next round of Software CQI.
    a. 2b. pre- and post-conditions
    b. 2f. black-box testing
    c. 2g. white-box testing
    d. 3d. detailed design
    e. 3e. design to code
    f. 3i. team skills
    g. 3h. non-functional properties (not covered)
    h. 2e. testing (not assessed)
    i. 2a. architectural styles (not met)

Appendix

CS 3195-report.docx (Daniel Mejia)
CS 3331-report-badreddin.docx (Omar Badreddin)
CS 3331-report-mondragon.docx (Oscar Mondragon)
CS 4310-reprot.docx, cs4310-outcome-assessment.xlsx (Elsa Tai Ramirez)
CS 4311-report.docx (Steve Roach), cs4311-outcomes-summary.docx (Salamah Salamah).

CQI: CS 3195, Spring 2020

Course Number: CS 3195
Course Title: Junior Professional Orientation
Course Instructor: Daniel Mejia

Description
Introduction to the Computer Science profession with a special emphasis on professional ethics. Required of all students prior to graduation.

Textbook
Baase, Sara (2018). *A Gift of Fire: Social, Legal, and Ethical Issues for Computing Technology (5th Edition)*

Grade Distribution
The class enrollment was 63 students at census day.

Grade	Pass				Fail
	A	B	C	D*	F
No. of students	49/63	11/63	0/63	1/63	2/63
Percentage	77.8%	17.4%	0%	1.6%	3.2%
Total	61/63 (96.8%)				2/63 (3.2%)

*D is a passing grade in CS 3195.

Out of 63 students who were enrolled in the class 61 students (96.8%) passed, with a D or greater; of those, one student (1.6%) passed the class by receiving a D. 2 students (3.2%) failed, including one who did not take the final exam. Of the failures, one student did not submit 8 assignments including 3 major assignments and the final exam; the student only attended 38% of the course meetings. The other student who failed missed 4 assignments, all of which were major class assignments.

Learning Outcomes
Assessment was done using reading quizzes, assignments, and exams.

For each outcome I measured the average score obtained by students on the quiz, assignment, or exam. The nominal acceptance value was 70%.

Learning outcomes
Level 1: Knowledge and Comprehension:
Level 1 outcomes are those in which the student has been exposed to the terms and concepts at a basic level and can supply basic definitions. Upon successful completion of this course, students will able to:

a. Describe techniques for face-to-face and telephone interviews.
   Mock Interview – 95.2%

Excellent

b. Recognize possible post-baccalaureate paths, including graduate study, entrepreneurship, and employment in government, academia, and the private sector.
Career Fair – 96.8%
Entrepreneurship Assessment – 89.7%
Job Research – 88.6%
Good

c. Describe the role of ethics in society and software engineering.
Ethics Report – 89.9%
Final Exam – 92%
Quiz 1 – 85.6%
Good

d. Describe the need and venues for continuing professional development.
Extended Goals & Deficiencies – 91%
Quiz 5 – 87.3%
Good

Level 2: Application and Analysis:
Level 2 outcomes are those in which the student can apply the material in familiar situations, e.g., can work a problem of familiar structure with minor changes in the details. Upon successful completion of this course, students will be able to:

a. Set short-term and long-term goals based on one's strengths, weaknesses, and experiences.
Goals & Planning Worksheet – 93%
Extended Goals & Deficiencies – 91%
Excellent

b. Prepare for and participate in a mockup interview.
Mock Interview – 95.2%
Excellent

c. Evaluate the impact of computer science solutions on individuals, organizations, and society.
Quiz 1 – 85.6%
Good

d. Prepare a portfolio that includes a cover letter, resume, samples of software development experiences, oral communication, and written communication samples
Online Portfolio – 70%
Resume – 85%
Mastering Dinner Etiquette – 81.3%
Satisfactory

Analysis of the Results, and Instructor's Recommendations

- The questions were mapped to the learning outcomes based on the relationship of the assignments to the learning outcomes. The averages indicate that the students who passed this class have "good" to "excellent" knowledge of all the necessary topics. A majority of the learning outcomes were also covered by lectures and guest speakers; it may be useful to create assignments based on speakers and the material that the covered

Recommendations

- Consider moving learning outcome 1.C to level 2
    - Rationale: It is critical for students to be able to synthesize their thoughts on ethics and their responsibilities they have in their career to ensure appropriate practice
- Add an additional level 2 outcome: "Prepare a written document expressing proper technical writing in professional settings, through email, written reports, and evaluation of products/services"
    - Provide students with an additional written assignment so that they can practice their technical writing skills and etiquette
    - Students should be able to clearly express their thoughts using appropriate grammar and style
- Expand on the use of technology for communication and collaboration
- Continue having student support in the form of an Instructional Assistant (IA)

Course Outcomes Analysis: CS 3331, Spring 2020

Number: CS 3331
Title: Advanced Object-Oriented Programming
Instructor: Omar Badreddin

Description
An in-depth exposure to the Object-Oriented Programming (OPP) paradigm, which builds upon programming experiences gained in lower-level computer science classes. Emphasis on programming in an object-oriented language with which students are already familiar with and on applying advanced OOP concepts to implement programming solutions. The course covers topics related to requirements, testing, code refactoring, and comprehension.

Textbooks
- Xiaoping Jia, *Object-Oriented Software Development Using Java*, second edition, Addison Wesley, 2002.
- Additional material is provided to students on Blackboard Learning System.

Grade Distribution
During this course offering, the instruction moved to virtual teaching around the middle of the semester (after Spring Break). As a result, the midterm exam grade was distributed to assignments and to the final exam. The students took the midterm as a take home exam, but the grade was not included in the final letter grade calculations.

Grade Distribution	%
Assignments 7 assignments, plus 1 warm-up and 1 optional	30
Quizzes (announced, in class)	10
Class attendance and participation	10
Midterm Exam	25
Final Exam	25

Final letter grade distributions are as follows.

Grade	Pass			Fail	
	A	B	C	D	F
No. of students (69)	22	14	8	2	6
Percent (%)	42%	26%	15%	4%	11%
Total	85% (61/69)			15% (8/69)	

Grade distribution in this semester was very similar to previous offerings.

Learning Outcomes
Assessment of the learning outcomes are based on 1) the final exam, and 2) the programming assignments. There were other student learning assessment tools, such as quizzes, midterm exam, in-class and virtual exercises, but they were not used for learning outcomes assessment.

Similar to previous assessments, I use 70% as the nominal acceptance value. For each learning outcome, I identify the exam question(s) and/or assignment question(s) that most closely map to the learning outcome. I then measure the average score representing the learning outcome. In calculating averages, I included entries where students did not submit an assignment.

Level 1: Knowledge and Comprehension
Level 1 outcomes are those in which the student has been exposed to the terms and concepts at a basic level and can supply basic definitions. The material has been presented only at a superficial level. Upon successful completion of this course, students will be able to:

1a. Explain the differences between an object-oriented approach and a procedural approach.
Satisfactory (Assignment 1 and Assignment 2, 100%)

Level 2: Application and Analysis
Level 2 outcomes are those in which the student can apply the material in familiar situations, e.g., can work a problem of familiar structure with minor changes in the details. Upon successful completion of this course, students will be able to:

2a. Formulate use-case diagrams and scenarios to support understanding of user requirements.
Satisfactory (Assignment 2, Final Exam Q1; 87%)

2b. Use object-oriented design notations, including UML class diagrams and state machine diagrams (optionally sequence diagrams) to model problem solutions.

I did not cover sequence diagrams in this offering, even though it was covered in previous offerings. This is because sequence diagrams are optional, and to help compensate for the lost week of lectures due to the pandemic.

Class Diagrams: **Satisfactory** (Assignment 3, Final Exam Q2; 79%)
State machines: **Satisfactory** (Assignment 4, Final Exam Q 4; 96%)

Overall, **Satisfactory** (87.5%)

2c. Use basic object-oriented design patterns to structure solutions to software design problems.
Design Patterns topic is covered towards the end of the semester. While there were several questions that addressed design patters in the exam, I use Assignment 6 to evaluate the learning outcome. This is because Assignment 6 focused on using and implementing several design patterns in Java.

Satisfactory (Assignment 6; 90%)

2d. Translate design features, such as classes and their relations, to implementations.
There were several assignments and in class exercises where students translate a design into an implementation. For this LO, I use assignments 3 and 4 where students had to develop a design and implement it. The design was represented in both, a class diagram and a state machine.

Satisfactory (Assignments 3 and 4; 87.5%)

2e. Use frameworks and library classes and methods in problem solutions.
The frameworks used in this offering included Java Swing, Java FX, Junit, and JavaDoc. Library classes were used in several assignments. To assess this Learning Outcome, I use students grades in assignment 5, which focused on Junit and JavaDoc.

Satisfactory (Assignment 5; 91%)

Level 3: Synthesis and Evaluation
Level 3 outcomes are those in which the student can apply the material in new situations. This is the highest level of mastery. Upon successful completion of this course, students will be able to:

3a. Design and implement software employing the principles of modularity, encapsulation, information hiding, abstraction, and polymorphism.
This is a central theme in this course. This LO was evaluated repeatedly in many assignments as well as in the final exam. I use Question 3 in the final exam to evaluate this learning outcome. This question required students to apply these concepts by refactoring existing code to enhance its modularity, encapsulation and information hiding, and abstraction using inheritance.

Satisfactory (Final exam Q3; 75%).

3b. Design, implement, and use classes and methods that follow conventions and styles, and make appropriate use of advanced features such as inheritance, exception handling, and generics.
For this LO, I use assignment 3 for evaluation.

Satisfactory (Assignment 3; 81%)

3c. Evaluate existing classes and software for the purposes of extension through inheritance.
This LO was most appropriately evaluated in the final exam Q3.

Satisfactory (Final exam Q3; 75%).

3d. Create API documents for classes, fields and methods.
This was most appropriately evaluated in Assignment 5 (part I), and in the final exam Q3.c.

Satisfactory (Assignment 5 part I, Final exam Q3.c; 87%)

3e. Design and implement test suites for automated unit testing.
This was most appropriately evaluated in Assignment 5 (part II), and in the final exam Q3.d.

Satisfactory (Assignment 5 part II, Final exam Q3.d; 77%)

3f. Re-factor existing source code to improve its design or efficiency.
In Assignment 6 part II, students had to refactor existing code to make use of design patterns.

Satisfactory (Assignment 6 Part II, 74%)

In summary, all course outcomes were met satisfactory.

Observations and Recommendations

1. The number of enrollments in this course went down from 70 students (in Fall 2019) to 50 students in this evaluation period (Spring 2020). My view is that students' learning has improved noticeably in part due to the smaller class size. This is particularly true when students must complete hands-on exercises in class using new frameworks (like Junit and JavaDoc for example). It was much more feasible to address issues for more students in class without significantly delaying the lecture when the class size is more manageable.

2. Instructional Assistance (IA, or Undergraduate Teaching Assistance) provides significant value for this course. IAs help students primarily during the lecture, but also outside the classroom. I strongly recommend continuing this IA support for CS 3331 in the future.

3. This course offering went virtual right after the Spring Break. Overall, and in my view, there was very little negative impact from going virtual. Before the closure of campus, we had prepared the Blackboard Collaborate Ultra and shared link and instructions with all students. I summarize the positive and negative impacts from virtual learning in the following points.

    - **Positive Impact:**

        1) **Lecture recordings** available to students who cannot attend at the scheduled time, and for those who need to review the lecture at a later time.

        2) **Introvert students** who may be shy in classroom sittings were much more likely to engage using chat. I have noticed several students who have become more actively engaged in discussions and solving problems during the lecture.

        3) **Course IA and TA** were available during the lecture and provided valuable and timely help to students during the lecture without creating interruptions (often by sending direct messages to students facing problems or have questions).

        4) **Common problems** and issues with platforms and frameworks were available to students who need them on-demand. As an example, TA and IA were able to copy/paste instructions to resolve issues with JavaDoc and Junit and send directly to students who need them privately, resulting in very little interruptions to lecture flow.

        5) **Automated Attendance.** Virtual lectures largely simplified attendance taking.

    - **Negative Impact:**

      Besides the expected loss of person-to-person communications and interactions, the following are three key negative impact from virtual learning.

        1) **Interactive Cooperative Design**. I could not find an appropriate replacement to the large whiteboard in the classroom to aid in teaching UML and design patterns in particular. Large whiteboard in the class allowed me to create imperfect and incomplete designs to motivate class-wide discussions. This required frequent editing to the design which was more time consuming using digital means.

        2) **Before and after lecture interactions**. Being available before and after the lecture was a good time for students to communicate concerns or get quick help with concepts or assignments. I managed to reduce this negative impact by creating a *Slack Time* before and after the lecture where I am available for students to answer questions.

        3) **Cooperative learning**. While this is difficult to assess, I have noticed that students in the classroom cooperate more effectively particularly on in-class exercises. I was not able to use breakout rooms effectively to replace this instantaneous cooperative learning.

4. The outcome 2b (UML) was revised to make the sequence diagram optional and focus more on other modeling notations (like state machines, use cases, and class diagrams). This revision was approved by the faculty. I had covered sequence diagrams in previous offerings but not during this evaluation period (Spring 2020). I have explained to students that we are only covering a subset of UML, and that there are several other modeling notations that are not covered in this course that may be worthy of exploring.

5. TA support is essential in this course. I have only required the TA to come to lectures where I anticipated significant hands-on exercises or introduction of new frameworks. This allowed the TA to put more effort in carefully and consistently grading assignments, especially that this offering included large number of assignments with several parts each.
6. I have frequently related to current events in the tech world, like the development of specific sophisticated software or prominent software failures to motivate several topics in this course. I find this to work extremely well in motivating learning.
7. I have adopted several learning paradigms in this course in effort to make each lecture unique to reduce fatigue. I find problem driven paradigm to be very effective in this course. Often, I introduce the problem and demonstrate it in code, before we discuss the underlying principle. This worked extremely well for Java Interfaces, Design Patterns, and several design notations including use cases, class diagrams, and state machines.
8. This offering was fortunate to have a TA who is very well prepared and has been the TA for the course for several years. In the past, it was very difficult to find a TA who is reliably knowledgeable particularly about specifics related to design notations. I strongly commend the effort to maintain the same course TA for several offerings. As a recommendation, extra care should be put to select the most appropriate and qualified TA for CS3331.
9. All assignments were done individually. While this created additional load for grading for me and the TA, I find it more effective in ensuring that all students practice applying concepts in their programming assignments.
10. The majority of the assignments were independent; i.e, they did not require the student to have completed the assignment before it. I find this to help students, particularly those who may fall behind in early assignments.
11. The final exam was a take home open notes open book exam. The final exam was lengthy and extensive to ensure that students do not have time to collaborate. The exam also included several design questions that require students have solid understanding of fundamental Object Oriented concepts. As a result, not all students were able to answer all questions due to time constraints. This resulted in relatively lower satisfaction rates for several learning outcomes.

Course Outcomes Analysis: CS 3331, Spring 2020

Number: CS 3331
Title: Advanced Object-Oriented Programming
Instructor: Oscar A. Mondragon

Description
An in-depth exposure to the object-oriented programming paradigm, which builds upon programming experience gained in lower-level computer science classes. Emphasis on programming in an object-oriented language with which students are already familiar, and on requirements, testing, code reading, and comprehension.

Textbooks
- Xiaoping Jia, *Object-Oriented Software Development Using Java*, second edition, Addison Wesley, 2002.

Recommended Books (Not required):
1. Head First Design Patterns. Eric Freeman and Elizabeth Freeman. O'Reilly 2004.

2. Head First Object-Oriented Analysis and Design. Brett D. McLaughlin, Gary Pollice, and Dave West. O'Reilly 2006.

3. The Elements of Java Style. Allan Vermeulen, et al. Cambridge University Press, 2000.

4. Martina Seidl, et al., UML@Classroom: An Introduction to Object-Oriented Modeling, Springer, 2015 ([e-book] through UTEP library)

5. Cay S. Horstmann, Core Java Volume I - Fundamentals, 11th edition, Prentice Hall, 2018 ([e-book] through UTEP library)

Grade Distribution
This term is unprecedent because of the COVID 19 pandemic and the impacts on course policies, instructional method, and students and faculty performance under such circumstances.

The class started with about 50+ students enrolled, and finished with 41 students. Several students dropped before the course withdrawal date, which was extended until the end of semester. The table below summarizes the distribution of final letter grades that the remaining 41 students earned.

Grade	Pass			Fail	
	A	B	C	D	F
No. of students (41)	9	12	10	6	4
Percent (%)	22%	29%	24%	3%	9%
Total	76% (31/41)			24% (10/41)	

Out of four students who received an F, three didn't take the final exam.

Learning Outcomes
Assessment was done with final exam, Labs, student presentation, and homework assignments. There were frequent BB quizzes upon reading assignments at the beginning of the classes, but they were not included for the assessment.

For each of the measured outcomes, the average score obtained by students on the assessments listed above mapped to the outcome, and the nominal acceptance value was 70%. Most outcomes had multiple

questions/assessments mapped to them, and for these outcomes, the most relevant or representative questions/assessments were selected.

Level 1: Knowledge and Comprehension
Level 1 outcomes are those in which the student has been exposed to the terms and concepts at a basic level and can supply basic definitions. The material has been presented only at a superficial level. Upon successful completion of this course, students will be able to:

1a. Explain the differences between an object-oriented approach and a procedural approach.
Satisfactory (77.7%: Lab 1: 79.8%, Lab 2: 77.9%, Lab 3: 75.5)

Level 2: Application and Analysis
Level 2 outcomes are those in which the student can apply the material in familiar situations, e.g., can work a problem of familiar structure with minor changes in the details. Upon successful completion of this course, students will be able to:

2a. Formulate use-case diagrams and scenarios to support understanding of user requirements.
Satisfactory (74%: Final Exam Q16: 67.2%, System Analysis with CRC Q2: 81%)

2b. Use object-oriented design notations, including UML class diagrams and state machine diagrams (optionally sequence diagrams) to model problem solutions.
Satisfactory (75.3%: Final Q19: 73.4%, System Analysis with CRC Q7: 76.2% Q8: 76.2%)

2c. Use basic object-oriented design patterns to structure solutions to software design problems.
Satisfactory (85%: Group Design Pattern Presentations: 93.6%, Lab 5: 76.4%)

2d. Translate design features, such as classes and their relations, to implementations.
Satisfactory (Lab 5: 76.4%)

2e. Use frameworks and library classes and methods in problem solutions.
Satisfactory (76.8%: Lab 5: 76.4%, Lab 4: 77.2%)

Level 3: Synthesis and Evaluation
Level 3 outcomes are those in which the student can apply the material in new situations. This is the highest level of mastery. Upon successful completion of this course, students will be able to:

3a. Design and implement software employing the principles of modularity, encapsulation, information hiding, abstraction, and polymorphism.
Satisfactory (Lab 3: 75.5%)

3b. Design, implement, and use classes and methods that follow conventions and styles, and make appropriate use of advanced features such as inheritance, exception handling, and generics.
Satisfactory (76.8%: Lab 5: 76.4%, Lab 4: 77.2%)

3c. Evaluate existing classes and software for the purposes of extension through inheritance.
Satisfactory (Lab 3: 75.5%)

3d. Create API documents for classes, fields and methods.
Not covered (Topic not covered, although students wrote some lab reports and documented design as design homework)

3e. Design and implement test suites for automated unit testing.

Satisfactory (Lab 5: 76.4%)

3f. Re-factor existing source code to improve its design or efficiency.
Satisfactory (76.8%: Lab 5: 76.4%, Lab 4: 77.2%)

In summary, all course outcomes were met satisfactory, but one not covered.

Observations

Because of the unprecedent circumstances of COVID 19, loosing one week and transition to remote teaching on the fly, two topics were not covered. In addition, some student struggle to adapt to remote teaching including remote office hours affecting their performance, specially in the Lab assignments.

1. The class was conducted as project base learning making use of a chess game system. The system was built incrementally and students had to refactor code from procedural to object oriented to design patterns. Students did a report for some labs.
2. The outcome 2b (UML) considers Use Case, Class, Sequence and State Machine diagrams.
3. The Class Responsibility Collaboration method was used to help students perform Object Oriented Analysis to identify potential classes and their relationships. CRC was taught after use cases and scenarios. The output of this method facilitated doing the class diagrams and sequence diagrams.
4. Due to the large class, each student demonstrated their labs twice with the support of both TA and IA.
5. I agree with last year observation: "Due to the nature of this course (low-level design and coding), adequate TA support is essential for its success. The course TA should be able to give good and effective feedback on students' design and coding work, be knowledgeable on OO design and programming, and be familiar with development environments (Eclipse/IntelliJ IDEA, Git, Junit)."
6. I also agree with last year observation: "The undergraduate instructional assistant was very helpful with a large class size. I noticed that many students prefer to see the IA for programming-related questions" as well as for office hours. They had very different styles.
7. Labs were use to make evident the limitations of procedural programming; overcome these limitations with classes, inheritance and polymorphism; and to enhance code with patterns and generalization.
8. Quizzes were delivered at the beginning of class to test reading assignments or review concepts covered in previous lectures.

Recommendations

1. The topic of API documentation (a L3 outcome) was not covered although students were asked to create reports of some labs to document their code and design of labs. Students were asked to document the chess game system using UML diagrams and CRC.
2. Learning outcome 2 e) was partially covered with Java collections and GUI; however, multithreading, and networking was not covered.
3. Identifying classes to solve problems is not trivial and students struggle to find them during object-oriented analysis. Consider using Class Responsibility Collaboration as a methodology to find classes after they have created scenarios for the use cases.
4. Consider to request the first attempt to solve part of the project class using a procedural approach so students can see the limitations of code reuse, inheritance and polymorphism.
5. Provide help in identifying a qualified TA who is knowledgeable on the course subject, e.g., someone who took this class and/or CS 4310/4311.

6. Ask for continued support for an undergraduate teaching assistant (IA) that has already taken this class.

CQI Assessment Report
CS 4310
Spring 2020

Department and Course Number: CS 4310 (27778)
Course Title: Software Engineering I: Requirements Engineering
Course Instructor: Elsa Tai Ramirez
Teaching Assistant: Benjamin Robertson

Course Overview: Software Engineering I: Requirements Engineering (CS4310) is the first semester of a two-semester capstone course in which students work with a customer to capture and specify requirements for a real-world application. The spring 2020 project required the class to work with a team from the Army Research Laboratory. The real-world application is called the Findings and Reporting Information Console (FRIC) that the Army uses during cyber engagements. FRIC allows the lead analyst to manage tasks and subtasks for a cyber engagement and monitor analysts' progress during the cyber engagement. It allows analysts to document potential issues and confirmed vulnerabilities and mitigations discovered during the cyber engagement. It also allows lead analyst and analyst to generate different types of reports.

On top of preparing students to become proficient in applying software requirements engineering methods and techniques, the students also learn how to work in cooperative teams and lead and manage a project. The students are assigned to teams of 5 with each team member playing a different software engineering role. The teams are formed based on the members' personalities, their role preferences, their experience, and their grade point averages. They learn team building and developing using PIGSFACE (P = Positive interdependence, I = Individual Accountability, G = Group Processing, S = Social Skills, and FACE = Face to Face Promotive Interaction). Each team member takes turns to lead a major deliverable to practice their leadership skill.

Major deliverables in this class include interview questions and memo, interview report, feasibility report, prototype, software requirements specification and models. Each team performs group processing after delivering the interview report, feasibility report, and prototype. The purpose of group processing is to reflect on how the team performed, what worked, and what did not worked. Teams also get feedback on their work, deliverables, and team performance from the guidance team on a regular basis. At the end of the semester, the teams present their work to the customers and the members of the guidance team.

Textbook: Dick, J, Hull, E., and Jackson, K. Requirements Engineering 4th Ed. 2017 Edition Springer (https://www.springer.com/us/book/9783319610726).

Course Goals: To prepare students to become proficient in applying software requirements engineering methods and techniques, working in cooperative teams, and managing projects. This course is also designed to promote your overall success, inside and outside the classroom.

CS 4310 Course Outcomes:

Level 1. Knowledge and Comprehension

Level 1 outcomes are those in which the student has been exposed to the terms and concepts at a basic level and can supply basic definitions. The material has been presented only at a superficial level. Upon successful completion of this course, students will be able to:

a. Define basic software engineering concepts and principles (abstraction, anticipation of change, modularity, stepwise refinement, and separation of concerns).
b. Define quality attributes such as availability, correctness, efficiency, interoperability, maintainability, portability, reliability, security, modifiability, availability, testability, and usability.
c. State the main features of process improvement models, e.g., CMM, ISO, PSP, QPI, Plan-Do-Check.
d. Define security design principles and the rule of least astonishment.

Level 2: Application and Analysis

Level 2 outcomes are those in which the student can apply the material in familiar situations, e.g., can work a problem of familiar structure with minor changes in the details. Upon successful completion of this course, students will be able to:

a. Determine which life cycle model to use by analyzing different scenarios.
b. Apply techniques for eliciting requirements.
c. Analyze requirements to determine if they meet the attributes of well-written requirements.
d. Identify risks in software development and project management.
e. Analyze the course project and determine the local and global impact on computing on individuals, organizations, and society, including consideration of professional software engineering code of ethics.
f. Relate the importance of professional societies.
g. Engage in self-directed study to learn new techniques and tools for software requirements definition.

Level 3: Synthesis and Evaluation

Level 3 outcomes are those in which the student can apply the material in new situations. This is the highest level of mastery. Upon successful completion of this course, students will be able to:

a. Construct a feasibility report that demonstrates ability to engage in self-directed study.
b. Conduct verification and validation using techniques such as inspections and walkthroughs.
c. Construct a prototype, which adheres to basic HCI principles and applicable security design principles, to validate the user interface.
d. Construct a software requirements specification.
e. Analyze and model aspects of a problem by applying various modeling techniques.
f. Demonstrate an ability to assemble and orally present technical work and compose technical documents that are grammatically correct and technically sound.
g. Apply effective techniques for project management, collaboration, and problem-solving within groups.

Summary of the Assessment of the Course Outcomes

The course assessment was based on the two midterm exams, a final exam, assignments, presentations, and a team project. The course project, which centered on the development of a Findings and Reporting Information Console, included the development of an interview report, feasibility study, prototype, software requirements specification document, and a traceability report. There were a number of in-class exercises. Exams, team documents, and exercises are available upon request.

	T1	T2	Final	Project	Assignment	Outcome
1a. Define basic software engineering concepts and principles	PII-1 (Total: 25; Mean: 16.63), PII-2 (Total: 25; Mean: 15.04), PII-2(Total: 25; Mean: 15.04)	Q3 (Total: 25; Mean: 21.21), Q4 (Total: 25; Mean: 22.45)	Q3a (Total: 2; Mean: 1.96), Q3b (Total: 2; Mean 1), Q3c (Total: 2; Mean: 1.96), Q3da (Total: 1; Mean: 0.08), Q3db (Total: 1; Mean 0.26), Q3dc (Total: 1; Mean: 0.28), Q4 (Total: 14; Mean: 10.42), Q5 (Total: 10; Mean: 9.67), Q6 (Total: 14; Mean: 10.94), Q7 (Total: 14; Mean: 11)	Feasibility Report (Total: 100; Mean: 78.50), SRS (Total: 100; Mean: 92), Final Presentation (Total: 100; Mean: 88.40)	In-class Exercise and Presentation	Good
1b. Define quality attributes.			Q1e (Total: 5.5; Mean: 2.82)	SRS (Total: 100; Mean: 92), Final Presentation (Total: 100; Mean: 88.40)	In-class Exercise and Presentation	Good
1c. State the main features of process improvement models.						This topic was not covered due to the time constraint associated with losing two classes.
1d. Define security design principles and the rule of least astonishment.		Q1a - 1d (Total: 25; Mean: 18.52)				Acceptable
2a. Determine which life cycle model to use by analyzing different scenarios.	Q5 (Total: 5; Mean: 2.52)		Q2a (Total: 2.5; Mean: 0.54), Q2b (Total: 2.5; Mean: 0.95)		In-Class Exercise and In-Class Quiz (Total: 100; Mean: 43)	Marginal
2b. Apply techniques for eliciting requirements.	Q2 (Total: 3; Mean: 1.64), Q3 (Total 6: Mean: 4.5), Q5 (Total: 5; Mean: 2.52), Q6 (Total: 4; Mean: 3.36)			Interview with clients; Presentations to clients (Use case Presentation; DFD Presentation; Class Diagram Presentation; State Transition Diagram; Prototype Presentation), Interview Memo and Questions (Total: 100; Mean: 90.20),		Good

				Interview Report (Total: 100; Mean: 82.70)		
2c. Analyze requirements to determine if they meet the attributes of well-written requirements.		Q2a - 2g (Total: 25; Mean: 17.37)	Q1a (Total: 5.5; Mean: 3.91), Q1b (Total: 5.5; Mean: 4.95), Q1c (Total: 5.5; Mean: 2.14), Q1d (Total: 5.5; Mean: 3.76), Q1e (Total: 5.5; Mean: 2.82)	SRS (Total: 100; Mean: 92), Final Presentation (Total: 100; Mean: 88.40)	In-Class Exercise and Presentation	Good
2d. Identify risks in software development and project management.	Q3 (Total 6: Mean: 4.5), Q7a (Total: 3; Mean: 2.94), Q7b (Total: 3; Mean: 2.88)			Team Work, Group processing of Interview Report, Feasibility Report, and Prototype.		Good
2e. Analyze the course project and determine the local and global impact on computing on individuals, organizations, and society, including consideration of professional software engineering code of ethics.						This topic was not covered due to the time constraint associated with losing two classes.
2f. Relate the importance of professional societies.						This topic was not covered due to the time constraint associated with losing two classes.
2g. Engage in self-directed study to learn new techniques and tools for software requirements definition.				Interview Memo and Questions (Total: 100; Mean: 90.20), Interview Report (Total: 100; Mean: 82.70), Feasibility Report (Total: 100; Mean: 78.50), Prototype (Total: 100; Mean: 80.88), SRS (Total: 100; Mean: 92),		Good

				Traceability (Mean: 100; Mean: 95)		
3a. Construct a feasibility report that demonstrates ability to engage in self-directed study.	Q9a (Total: 2; Mean: 1.48), Q9b (Total: 4; Mean: 2.42)			Feasibility Report (Total: 100; Mean: 78.50)	In-Class Exercise and Presentation	Acceptable
3b. Conduct verification and validation using techniques such as inspections and walkthroughs.	Q1a (Total: 2; Mean: 1.36), Q1b (Total: 2; Mean: 1.12), Q8a (Total: 2; Mean: 1.76), Q8b (Total: 2; Mean: 1.7), Q8c (Total: 2; Mean: 1.66), Q10 (Total: 4: Mean: 1.2)			Presentations to clients (Use case Presentation; DFD Presentation; Class Diagram Presentation; State Transition Diagram; Prototype Presentation), Traceability (Mean: 100; Mean: 95), Final Presentation (Total: 100; Mean: 88.40)	In-Class Exercise and Presentation	Good
3c. Construct a prototype, which adheres to basic HCI principles and applicable security design principles, to validate the user interface.		Q1a - 1d (Total: 25; Mean: 18.52)		Prototype (Total: 100; Mean: 80.88)		Good
3d. Construct a software requirements specification.				SRS (Total: 100; Mean: 92)		Good
3e. Analyze and model aspects of a problem by applying various modeling techniques.	PII-1 (Total: 25; Mean: 16.63), PII-2 (Total: 25; Mean: 15.04), PII-2(Total: 25; Mean: 15.04)	Q3 (Total: 25; Mean: 21.21), Q4 (Total: 25; Mean: 22.45)	Q4 (Total: 14; Mean: 10.42), Q5 (Total: 10; Mean: 9.67), Q6 (Total: 14; Mean: 10.94), Q7 (Total: 14; Mean: 11)	Feasibility Report (Total: 100; Mean: 78.50), SRS (Total: 100; Mean: 92), Final Presentation (Total: 100; Mean: 88.40)	In-Class Exercise and Presentation	Acceptable
3f. Demonstrate an ability to assemble and orally present technical work and compose technical				Final Presentation (Total: 100; Mean: 88.40); Interview with clients; Presentations to clients (Use case Presentation; DFD Presentation;		Good

documents that are grammatically correct and technically sound.				Class Diagram Presentation; State Transition Diagram; Prototype Presentation), Interview Memo and Questions (Total: 100; Mean: 90.20), Interview Report (Total: 100; Mean: 82.70), Feasibility Report (Total: 100; Mean: 78.50), Prototype (Total: 100; Mean: 80.88), SRS (Total: 100; Mean: 92), Traceability (Mean: 100; Mean: 95)		
3g. Apply effective techniques for project management, collaboration, and problem-solving within groups.	Q4a (Total: 4; Mean: 2.68), Q4b (Total: 2; Mean: 1.78), Q7a (Total: 3; Mean: 2.94), Q4b (Total: 3; Mean: 2.88)			Final Presentation (Total: 100; Mean: 88.40); Interview with clients; Presentations to clients (Use case Presentation; DFD Presentation; Class Diagram Presentation; State Transition Diagram; Prototype Presentation), Interview Memo and Questions (Total: 100; Mean: 90.20), Interview Report (Total: 100; Mean: 82.70), Feasibility Report (Total: 100; Mean: 78.50), Prototype (Total: 100; Mean: 80.88), SRS (Total: 100; Mean: 92), Traceability (Mean: 100; Mean: 95)	Team Work, Group processing for Interview Report, Feasibility Report, and Prototype.	Good

Grade Distribution

50 students were enrolled in this course with the following distribution (some students selected S/U option. On his/her transcription, instead of a letter grade, a S is displayed if he/she received a C or above.):

Grade	A	B	C	D	F
	4/50	23/50	19/50	4/50	0/50
Percent	8%	46%	38%	8%	0%

Summary

Overall, the outcomes were met, with the exception of outcomes 1c, 2e, and 2f. Outcomes 1c, 2e, and 2f were not covered because of the time constraint associated with losing two classes. Topics associated to outcome 2a should be introduced earlier on during the semester.

Recommendations

The recommendations from the 2018 report and how they were addressed are as follows:

- Introduce Software Engineering Code of Ethics earlier in the course, and add more questions on the exams: This topic was skipped because of the time constraint associated with losing two classes. I made the assumption that all students are required to pass CS 3195 and the Code of Ethics was covered in that class.
- Include a discussion and self-reflection exercise on professional societies (Outcome 2f). This could be accomplished through an essay assigned in the class after the lecture, or include it in the Social Impacts exercise: This topic was skipped because of the time constraint associated with losing two classes. This topic will be covered in the Fall 2020 if the committee decides to keep this outcome.
- Expose students to the practice of assessing risk of software and software development in the fundamental classes: Students were required to interact with the clients directly on 7 different occasions as compared to 3 in previous semesters. On top of the interview with clients, prototype presentation, final presentation, each team was required to present their models (use case diagram, class diagram, data flow diagram, and state transition diagram). The increase number of interactions allowed the students to understand the risk of incomplete, misunderstood, and missing requirements and the impact it has on the software project.
- If there is funding available to support a faculty member, create a one-credit hour course focused on non-functional requirements (carry-over from previous report): This was not addressed during the Spring 2020 semester.

The 2020 recommendations include:

- Introduce Software Engineering lifecycle earlier in the course.
- Expose students to the practice of assessing risk of software, software development, and team environment in the fundamental classes.
- Include a discussion and self-reflection exercise on professional societies (Outcome 2f). This could be accomplished through an essay assigned in the class after the lecture, or include it in the Social Impacts exercise (carry-over from previous report). This topic will be covered in the Fall 2020 if the committee decides to keep this outcome.
- If there is funding available to support a faculty member, create a one-credit hour course focused on non-functional requirements (carry-over from previous report).
- Use of Google Classroom/Docs/Sheets/Slides/Jamboard is effective in real time learning and team collaboration work.

	Learning Outcome	Cognitive Level
L1	a. Define basic software engineering concepts and principles (abstraction, anticipation of change, modularity, stepwise refinement, and separation of concerns).	Remember
L1	b. Define quality attributes	Remember
L1	c. State the main features of process improvement models, e.g., CMM, ISO, PSP, QPI, Plan-Do-Check.	Remember
L1	d. Define security design principles and the rule of least astonishment.	Remember
L2	a. Determine which life cycle model to use by analyzing different scenarios.	Apply
L2	b. Apply techniques for eliciting requirements.	Apply
L2	c. Analyze requirements to determine if they meet the attributes of well-written requirements.	Analyze
L2	d. Identify risks in software development and project management.	Understand
L2	e. Analyze the course project and determine the local and global impact on computing on individuals, organizations, and society, including consideration of professional software engineering code of ethics.	Analyze
L2	f. Relate the importance of professional societies.	Remember
L2	g. Engage in self-directed study to learn new techniques and tools for software requirements definition.	Apply?
L3	a. Construct a feasibility report that demonstrates ability to engage in self-directed study.	Create
L3	b. Conduct verification and validation using techniques such as inspections and walkthroughs.	Evaluate
L3	c. Construct a prototype, which adheres to basic HCI principles and applicable security design principles, to validate the user interface.	Create
L3	d. Construct a software requirements specification.	Create
L3	e. Analyze and model aspects of a problem by applying various modeling techniques.	Analyze

L3	f. Demonstrate an ability to assemble and orally present technical work and compose technical documents that are grammatically correct and technically sound.	Apply
L3	g. Apply effective techniques for project management, collaboration, and problem-solving within groups.	Apply

Proposed Cognitive Level	Proposed Changes
Apply	Apply basic software engineering concepts and principles (abstraction, anticipation of change, modularity, stepwise refinement, and separation of concerns). This should be a level 3 outcome instead of level 1.
Apply	Apply quality attributes. This should be a level 3 outcome instead of level 1.
Apply	Demonstrate how security design principles and rule of least astonishment can be used. This should be a level 3 instead of level 1.
Apply	Demonstrate how to address risks in requirements engineering and project management. Should this be a level 3 outcome?
	How come this learning outcome is a level 2? How come this is a learning outcome for CS 4310 and not CS 3195?
	Is it possible to rephrase this learning outcome to something more direct/concrete?
Create	New L3 learning outcome: Conduct an interview and construct an interview report.

Justification
We require the students to understand the concepts and apply the concepts to the project.
We require the students to understand the concepts and apply the concepts to the project.
We require the students to understand the concepts and apply the concepts to the project.
Students are asked to identifiy what the risks are, come up with migitations, and apply them.

Contents

Summary

Learning Outcomes	**Assessment**	Assessment
Level 1: Knowledge and Comprehension		
a) Articulate design principles, including cohesion and coupling, encapsulation, and information hiding.		Outcome is met
b) Describe software design concerns related to maintenance.		Outcome is met
c) Describe different software architectural styles (distributed, cloud, blackboard, event systems, layered system, and pipe and filters).	Lecture presentations	This outcome met
Level 2: Application and Analysis		
a) Identify appropriate software architectural styles (specifically distributed and cloud) given a software design problem.		This outcome is not met. We showed architectures, have an idea how they work, but don't really have students apply patterns to new situations.
b) Apply assertion-based techniques (such as pre- and post-conditions) to specify behavior of program modules.	Exam and project SDD	outcome partially met

c) Distinguish between the different levels of cohesion and coupling.	Exam	This outcome is met.
d) Use software development and maintenance tools, such as software documents creation and editing tools, GUI generators, comprehension and analysis tools, supporting activities tools (configuration management tools), verification and validation tools, and security vulnerability analysis tools.	Project	This outcome is partially met. Weakness in security and V&V tools.
e) Describe differences between unit, integration, system, and acceptance testing.	Exam	Not assessed directly. However, we have some evidence this outcome is met.
f) Apply black testing techniques to develop test cases for a variety of test coverages.	Project deliverable	This outcome is partially met. Weakness: Students had a hard time identifying classes and boundaries, in particular boundaries that are significant.
g) Apply white-box testing techniques to develop test cases for a variety of test coverages.	Project deliverable Exam	This outcome is partially met. The vast improvement would be to get students to recognize that a test case requires inputs and analysis of outputs.
h) Apply static and dynamic techniques to analyze non-functional properties, including common security vulnerabilities such as password weakness, over/underflows, and race conditions.	This outcome was not covered and was not assessed.	Not met.
i) Engage in self-directed study to learn new techniques and tools for software design, implementation, and/or testing.	Project deliverable	This outcome is met.
Level 3: Synthesis and Evaluation		
a) Conduct a technical review of software design, implementation, and V&V.	Project deliverable	This outcome was not measured.

b) Create and implement a software configuration management plan.	Project deliverable	This outcome is met.
c) Make use of appropriate architectural styles and diagramming techniques to define an architecture.	See discussion of 1c.	This outcome is met.
d) Make use of appropriate textual notations and diagramming techniques to create a detailed design for a software system.	Project deliverable	Outcome partially met. Needs improvement. SDDs are, in general, weaker than we'd like to see. It's difficult to manage generation of an SDD for the entire project. This might be better served generating the SDD for a subset of the project.
e) Construct software from a detailed design.	Project deliverable	This outcome should be reconsidered. It is atypical to have a detailed design from which to build software. It is more typical to refactor. Perhaps some combination of detailed design and iterative design/implementation would be reasonable. For example, do iterative for part of the project and up-front detailed design for another part. This outcome is partially met, if we accept refactoring as implementing to a design.
f) Develop a test plan for a software system.	Project deliverable	met
g) Demonstrate an ability to orally present a software design and implementation.	Project deliverable	met
h) Compose software design-related documents that are grammatically correct and technically sound.	Project deliverable	met
i) Apply effective techniques for collaboration and problem-solving within a team.	Not measured formally. Informally	Outcome partially met. While there is evidence that most students are able to function adequately

	assessed during team meetings.	in a team setting, it is not clear how effective this course is in promoting that.

Outcome 1a: Articulate design principles, including cohesion and coupling, encapsulation, and information hiding.

Assessment: Exam Question (expected answers in italics).

1. (40 points, 5 points each) Short Answers:

A) Briefly explain why content and common coupling are considered undesirable.

These provide uncontrolled access and visibility to data: it prevents encapsulation and data hiding, exposes implementation details, makes debugging difficult

B) Explain the differences between a walkthrough and an inspection.

Inspection: looking for specific faults. Walkthrough: looking for general acceptability of product.

C) Explain why (at least in some cases) inspections and walkthroughs are more effective than testing.

Can use higher-level reasoning.

D) Briefly explain why SCM is important for systems being developed by teams of developers. (Hint: You and I both know what SCM is. Tell me why you need it.)

Gives all team members access to most recent version of software. Keeps team members from colliding. Reproduce production builds.

E) Describe the primary differences between software engineering standards and standards in other engineering disciplines.

SE are process based, others are product based.

F) What is SWEBOK? Why is it useful?

Body of knowledge that helps define SE, se curricula, and se certification.

G) Regression testing is by definition the re-execution of previously successful tests. This sounds like a waste of time to run tests that have already passed. Explain why this might be useful.

Retest after refactor or any other change.

H) Describe one non-testing approach to program verification or validation.

Inspection, theorem proving, model checking, …

Student Examples:

Good:

A) The reason content and common coupling are considered undesirable is components are tightly coupling by means of accessing (sharing) global data. This means there may be unexpected results for those sharing the data if changed by just one component.
B) Walkthrough is more of an internal analysis of development members led by the **author** to analyze the product based on requirements and intended design. An inspection is a **more planned** approach in which a meeting is called, and **moderator** is brought in to analyze the product, each member has a role in evaluating the meeting and defects are detected.

C) It may be more of an effective form of testing as they are more of a preventative measure of analysis, that is code is not executed and done before system integration. This can lead to finding defects at an earlier stage and resulting in a more cost-effective approach.
D) SCM is important because it is a documented management guide for members in the handling of project's configuration items. Over time the system's configuration state changes, it is a well written SCM that is set to handle the ongoing changes to establish maintainability and traceability of such configurations.
E) Primary differences between SWE and other engineering disciplines are product evaluation standards are more defined for other engineering disciplines, not the case for SWE. Most SWE techniques cannot be scientifically justified, other engineering disciplines have some form of proof a technique will work.
F) A guide that covers effective SWE principles i.e. requirements, design, testing etc. as well as additional materials such as training resources. It is useful for SWE as it provides a means of resources that can promote an engineer's career profession.
G) It is useful to re-execute tests to account for changes that may affect other parts of the system. If this process is not done it may be harder to find defects as changes become harder to track.
H)Analysis is a non-testing technique that can evaluate classes over a singular test case, done on source code or model abstraction. The technique is not perfect and can fail if not carried out properly as in the case of a poor inspection.

Middle

A) It is considered a poor design choice because they both lack a clear responsibility for each class and it is difficult to determine which component affects a data element.
B) The differences are that a walkthrough is scheduled to be brief, the participants are the team members that is completing the project, finds anomalies and evaluate conformance, and can be used to identify errors in artifacts. An inspection is used to identify specific errors or problems, evaluate material with a specific aspect in mind, provides a list of items that must be present, the inspector is not part of the team making the project, is unbiased, does not invite management if it causes conflict, raises issues but does not solve them, records the issues in public, and sticks to technical issues.
C) Inspections and walkthroughs can be more effective in some cases than testing in that testing can be done by one person within the team with one point of view and may not see certain problems with what they are testing. In inspections and walkthroughs, they tend to include the whole team and provide more general coverage of the material. The different point of views from other members may in some cases identify a problem that wasn't apparent before. Additionally, having the point of view from someone external to the project can also provide to be more effective because they are not familiar with the project and can give you input on something one might have overlooked.
D) Helps to follow a version control system for the code within the team that facilitates changes in a controlled manner. It also helps by maintaining a history of the changes to each artifact which can be very useful. This is important because it allows the team of developers to have access to the code and make updates to the code without affecting the code that was already tested and functional. This updated code can later be merged to the main code in the master branch when it has been tested and accepted by either management, the lead, or the whole team.
E) The differences are that in software engineering standards there are guidelines for process and guidelines for techniques and a few in product evaluation while in other engineering standards they have guidelines for acceptable outcomes and guidelines for product evaluation. While both software and other engineering standards share guidelines for product evaluation, software engineering doesn't have as many. Software engineering standards focuses more on the process standards in comparison to manufacturing engineering.
F) It is a guide to the software engineering body of knowledge. It is useful because it helps to understand the foundations and as an emerging discipline it describes the accepted collective knowledge that one should master for software engineering. It covers anything from testing, maintenance, meeting software requirements, documentations like SRS, validation, verification, design, SCM, models, process, quality and practice.

G) It is useful because sometimes enhancements and refactoring one are of the code may change the previously working code. For example, if a superclass is modified then all subclasses must be tested even if they didn't change because they might have been affected.
H) One approach can be to trace the requirements of a system to the system design to make sure that there aren't any requirements misinterpreted or missing. This approach does not involve testing and is one approach to program validation.

Poor
A) Data responsibility isn't clear. Makes it hard to tell what code is affecting the data.
B) Walkthrough is more of a presentation of the product to get feedback from the clients. Inspection is a detailed analysis of the product to find defects, and ensure the product meets the requirements.
C) Its a different way of going through the product with outside eyes, which can allow you to see problems that are present that may have been neglected or overlooked.
D) SCM is used to monitor all changes in the product, allowing each team to be on the same page during any phase of development.
E) Software engineering is just the software portion of engineering. It's their one focus. Other engineering disciplines usually have some knowledge of other engineering disciplines.
F) It is the outline of software engineering. Its used to set the standard of software engineering, keeping it consistent.
G) Ensure changes haven't propagated unintended side effects.
H) Observation of the product. Observing the UI to ensure it meets the requirements.

Analysis

On this exam, the scores ranged from 40 down to 17. The average was 31, which is roughly 4/5 points for each answer.

The outcome is met.

Outcome 1b) Describe software design concerns related to maintenance.

Assessment: Exam question (Expected answers in italics)

5. (5 points) Between the solution in Question 3 and the solution in Question 4, which is more maintainable and why?
Clearly Q4.
Arguments:
Easier to extend or modify by just adding new sub-classes

No need to modify all following code segments when adding a step
Less global data: so objects have clear responsibility for data
Less non-essential data shared among objects
Each step is easier to find and understand (less code to read to get to it.)

Student Examples

I did not make copies of this exam, and do not have examples of student work.

Analysis

The scores ranged from 5 down to zero. The average was 4.1. The grading was 2 points for selecting the correct solution and 3 points for articulating at least one of the arguments.

Nearly 90% of the students selected the correct solution. Students had a harder time explaining why.

This is from the first exam. Had I repeated this type of question on the final, I would expect the explanations to improve, since this is practiced during lecture frequently.

This outcome is met.

Outcome 1c) Describe different software architectural styles (distributed, cloud, blackboard, event systems, layered system, and pipe and filters).

Assessment: Lecture Presentations/Exam

Students developed architecture lectures for a specific architecture in teams. These were presented to the class in lecture.

Exam questions, with correct answer underlined.

10. (2 points) Which of the following best describes strengths of plug-in architecture?

a. Allows developers to add functionality without modifying source code
b. Allows users to easily customize features (add or remove features)
c. Third parties develop features in collaboration with the original developer
d. Allows for isolation when solving problems
e. All of the above
f. a, b and d
g. a, c, and d

11. (2 points) Which of the following is an important aspect of event driven architecture?
    a. Scheduling a time to process events beforehand
    b. Event objects pass information on how to perform the process
    c. Every stage of event processing is performed on arrival
    d. All of the above
    e. None of the above

12. (2 points) True or False: one common connector in cloud architecture is the Internet.

13. (2 points) What are some applications of the Cloud Computing architecture?
    a. Enhance and implement backup and disaster recovery.
    b. Host and develop web and mobile apps.
    c. Distribute and supplement active directory.
    d. Test and development.
    e. All of the above
    f. None of the above

14. (2 points) An event driven architecture is best represented by:
    a. Highly scalable, decentralized, high latency
    b. Highly scalable, decentralized, low latency
    c. Highly scalable, centralized, high latency
    d. All of the above
    e. None of the above

15. (2 points) What are the 3 tiers in 3-Tier architecture?
    a. Persistence, middleware, back-end
    b. GUI, middleware, back-end
    c. Presentation, application, data

d. Model, view, control
e. All of the above
f. None of the above

16. (2 points) Which of the following best describes strengths and weaknesses of a Shared Nothing Architecture?
    a. Eliminates single point of failure, but costs more
    b. Nodes operate independently by ensuring memory coherence
    c. Allows each node to self-heal, but favors resilience over consistency
    d. Allows each node to self-heal, but requires system shutdown to install updates
    e. a and b.
    f. a and c.
    g. All of the above
    h. None of the above

17. (2 points) One of the main features of a RESTful system is:
    a. All data is encapsulated within and hidden by the processing components
    b. The interface is primarily through JavaScript, a well-known standard
    c. Systems are stateless - server does not need to know anything about the state of the client
    d. Representations capture the state of a resource and are transferred between components
    e. a and b.
    f. c and d.
    g. All of the above
    h. None of the above

18. (2 points) Why is there additional security risk associated with using a plug-in architecture?
    a. Plug-ins execute on the host system, and have access to everything the core system has access to
    b. Since the host can sanitize input, there is no additional security risk
    c. Since users must authenticate with the host, there is no additional security risk
    d. b and c.
    e. All of the above.
    f. None of the above.

19. (2 points) What is the difference between shared memory and shared nothing?
    a. In shared nothing, each node contains its own memory. In shared memory, each system uses the same memory space. In both, other resources such as disk may be used in common, for example in a multi-user RAID configuration.

b. In shared nothing, each node contains its own disk space and its own memory. Nodes do not share disk space or memory with other nodes. Whereas, in shared memory, nodes are sharing memory.

Analysis

The maximum score on this portion of the exam was 16. The average was 12.1. The minimum was 6. The lectures provided showed reasonable understanding of a single architecture.

Students have a basic understanding of architectures, and sufficient for them to describe architectural patterns.

This outcome is met.

Outcome 2a) Identify appropriate software architectural styles (specifically distributed and cloud) given a software design problem.

Assessment: See previous

Analysis

While students have a basic understanding, they are unlikely to have the experience to be able to select an architecture for a given problem or apply architectural patterns to new situations.

This outcome not met.

Outcome 2b) Apply assertion-based techniques (such as pre- and post-conditions) to specify behavior of program modules.

Assessment: Exam and Project SDD

(10 points) A simple calculator has a function that plots a line given its slope and intercept. The requirement reads, “The calculator shall accept as input an equation of the form Ax + By + C = 0 and plot the line where x is in [-4, 4]. The calculator should also accept two points on a line and generate the same plot.” The system implementation of this requirement is the class LinePlot which has the function LINE (A, B, C), where A, B, and C are the coefficients of the linear equation, and the function LINE(x1, y1, x2, y2), where the points (x1,y2) and (x2,y2) are on the line to be plotted. The output is a plot, as shown below for LINE(2, -1, 5).

The contracts for this class are:

Class: LinePlot

Super class: Plot

Description: Generate a line plot near the y intercept

Private Responsibilities:

Generate A,B,C coefficients from two points

Contracts:	Collaborations
LP1: Get A,B,C coefficients from user	PlotMenu
LP2: Get p1, p2 points from user	PlotMenu
LP3: Generate a Graphic object that DisplayScreen, Graphic represents a line plot from x=-4 to x=4	

if possible, or for x=x1+/-2, y=+/-2

otherwise

A) (10 points) Given this set of contracts, write the protocol for this requirement. Assume all of the values are double.

We have three contracts, which are things this class needs to support. There are lots of ways to split things, but let's start by supporting the three contracts. LP1 and LP2 can be combined.

- *Signature GetCoefficients() returns double A, double B, double C*
- *Purpose: Collect the coefficients of the line described by Ax+By+C=0. Either collect these from the user directly, or generate these from the input of two points (x1, y1) and (x2,y2), which are points on a line.*
- *Pre-conditions:*
  - *At most one of A, B, C can be zero.*
  - *(x1,y1) cannot be the same as (x2,y2) // require*
- *Post-conditions: A, B, and C are acquired from the user input // ensure*

Or an input error is signaled

Now for the generation of the plot:

- *Signature*
- *PlotLine(double A, double B, double C) returns Graphic*

Graphic is an object that can be displayed by DisplayScreen

- *Purpose: PlotLine generates a plot, in the region of the y intercept if it exists or is in the region of the x-intercept otherwise; This acquires coefficients A, B, and C are the coefficients of Ax+By+C=0 describing a line.*
- *Pre-conditions:*
- *A, B, and C are coefficients of the line described by Ax+By+C=0. At most one of A, B, C can be zero. //require*
- *Post-conditions A Graphic object containing the plot is created // ensure*

Acceptable would be

- *Signature*
- *PlotLine(double A, double B, double C) returns Graphic*
- *PlotLine(double x1, double y1, double x2, double y2) returns Graphic*

I wanted to see that you had 4 parts:

the signature with appropriate types

some description

preconditions
postconditions
The point is that I should be able to start coding based on the protocol.
If you specified some input parameters, I expected to see preconditions for them. What are ABC?
+4 if you gave me some protocol with the 4 parts clearly defined.
-5 if you didn't give me a signature
-3 for no pre and post
-1 for wrong return types
Varying credit based on how hard it was for me to figure out what you intended.
Makeup:
Had to see support for all three contracts.
8 pts if just LP3, but gets inputs.
At most 9 if inputs described but no signature

Analysis

On the exam, the maximum grade was 10, the average 6.2 and the minimum 0. I gave students the opportunity to resubmit exam questions with the opportunity to gain up to 50% of the points lost. After regrade, the average was 7.7.

In the exam, many students did well, but the average was sub-par. For this exam, I returned the exam and had them re-answer questions. On the makeup, the majority of students who gave initially poor answers greatly improved.

On the SDD, students struggle with pre and post conditions, in particular post conditions. We often see method post condition of "none", indicating at least some students do not understand what the post conditions are for.

This outcome is partially met

Outcome 2c) Distinguish between the different levels of cohesion and coupling.

Assessment Exam

2. (10 points) A system provides a variety of customer support services. Services are processed by two subsystems. One subsystem handles customer accounting, and the other handles actual performance of the support service. The data required by each of these subsystems is shown below, where the destination is marked by "A" for accounting, "S" for service, and "B" for both. A director process receives requests, builds the service data structure, and sends the data structure to the accounting subsystem and the support subsystem. Fragments are shown below.

```
typedef struct Service_struct {
  int service_id;            // B
  char cust_name[100];  // A
  char cust_addr[150];   // A
  char svc_type[20];      // S
  char[][] svc_params;   // S
} Service;

Director::main() {
    Service svc;
    svc = create_service();
    Accounting.process(svc);
    Support(svc);
}
```

(a) What type of coupling is this?

Stamp: passing data structure

(b) Describe an approach to reduce the coupling. What type of coupling are you suggesting? Why might this reduced coupling be better?

Pass the actual data instead of this structure; Data coupling)

Analysis

On this exam, scores ranged from 0 to 10. The class average was 7.6.
In this exam question, students had to identify the type of coupling/cohesion and find an appropriate approach to improve the design.

Many student struggle to identify the exact type of coupling/cohesion, but most of them are able to reduce coupling and increase cohesion.

This outcome is met.

Outcome 2 d) Use software development and maintenance tools, such as software documents creation and editing tools, GUI generators, comprehension and analysis tools, supporting activities tools (configuration management tools), verification and validation tools, and security vulnerability analysis tools.

Assessment

Assessment is based on the course project. Their success at this is indicated by their success in project.

Analysis

On the final exam, I ask students to name tools they would use on their next project. All students are able to identify useful development tools. For the project, all student teams used git for their repositories, and all used modern editors and GUI generators. Fewer used review and defect tracking tools. Very few used automated test tools.

This outcome is partially met, since security vulnerabilities and V&V tools are not covered.

Outcome 2e) Describe differences between unit, integration, system, and acceptance testing.

Assessment

I didn't assess this directly. I asked many questions of the type "design unit tests for ...." and "what system level tests are most significant for ...". Students can't get credit for answers is they give the wrong level of testing.

Analysis

Not assessed directly. However, we have some evidence this outcome is met.

Outcome 2f) Apply black testing techniques to develop test cases for a variety of test coverages.

Assessment Exam

2B (15 points) Create a unit test plan for the LINE function. Identify the test cases that you would use (you do not need to draw the output provided you explain what you expect to see). Explain the strategy you are using and justify your approach. (That is, tell me the strategy or process you are using to come up with inputs for a test set and convince me that it is appropriate for this problem.)

You guys didn't do well here. What I expected was boundary value or equivalence class. A number of you mentioned this, then failed to give me the boundaries or the classes.
We are plotting lines. You've been doing this since 8th grade. What kinds of lines do we have?

- *Horizontal*
- *Vertical*
- *Positive slope*
- *Negative slope*

These would be some equivalence classes you could have mentioned. It's the first set of things I would have tried. Some other types of classes I would have looked for are

- *Points in each of the 4 quadrants (16 cases of point pairs)*
- *Positive Y intercepts*
- *Negative Y intercepts*
- *Large Y intercepts*
- *Large negative Y intercepts*
- *Nearly vertical lines*
- *Vertical lines in -4<=x<=4*
- *Vertical lines where x<-4*
- *Vertical lines where x>-4*

Boundaries: Lots of you said [-4,4] was a range, thus +/-4 was a boundary. But you mis-used it. This is a range of x values for the output plot when the line is not vertical. It is not a limit on the input. Limits on the input are [-MAX, +MAX] for A, B, C, x1, x2, y1, y2. Boundaries I would consider would be -MAX, -1, 0, 1, +MAX. For really robust testing, I would try things like 1/MAX.

12 if you identified max, min and zero as boundaries, said BVA,
10 if you said BVA and gave reasonable examples
8 for BVA with no examples
6 for BVA and weird stuff (wrong things such as (0,0,0) is a horizontal line)

8 for a set of 4 or more reasonable input sets with no strategy
A tad more if you used max, 0, and min
4 for checking valid and invalid input but nothing else
If you said Line(0,0,0) is horizontal or vertical or true, I took off points.
If you gave me tests where no line is drawn and said there'd be one, I took off points.
If you gave me white box (e.g., data flow), I took off points.
If you gave me integration strategies (e.g., big bang) I took off points.
The more you said that was silly, the fewer points you received.
Regrade:
5 for saying BV and not giving me any real ones.
No credit for saying +/- 4 is an input restriction (i.e., that doesn't count as boundary)
However, if they actually used +/-4 to generate real BV tests, +12

10 for saying bv and ec, then only giving me ecs.

In general, you needed to tell me something like boundary value and here are the boundaries or equiv class and here are the classes. You should have indicated the expected output. If you said bv and gave me classes, no full credit. If you said classes and didn't identify them, no credit.

If you got less credit the second time, you lost points on the question. I said I wanted better answers. I'm not accepting crap.

Analysis

The exam grades ranged from 0 to 15. The initial average was 6.1. Students were given the opportunity to recover up to 50% of the points lost. After this second attempt, the class average was 8.5. They can say equivalence class and boundary value analysis, but they show difficulty in developing tests based on these concepts. Some students who gave initially poor answers greatly improved. However, the final average was only somewhat above 55%.

This outcome is partially met.

Outcome 2g) Apply white-box testing techniques to develop test cases for a variety of test coverages.

Assessment Exam

2. (25 points total) Consider the following Java code used to compute ....

2A) Write the control flow graph for this code. Use the letters at the left to label the nodes.

2B) Write a minimal test set that achieves statement coverage.

I'm looking for input. To get statement, you have to input two positive grades and a negative grade
What I want is two inputs and an output.
I'll give them 4 credit if they give me inputs w/o outputs
They might say EFG H I JK M N H P half credit.

2C) Write a minimal test set that achieves edge coverage.

Same test as above. You have to have two passes to hit all the statements, and this traverses all the edges

2D) Identify the error in the code and give a test case that will discover it.

Same test as above will discover it.
Error is line L needs to increment grade counter when there's input.

2E) What V&V technique did you use to identify the errors when answering question D?

Inspection, review, walkthrough. Zero if they say "testing, white box, black box"

Analysis

The scores ranged from 3 to 16. On this question, over 90% of the students can draw a correct CFG. Most students were able to show paths that achieve the various coverages. The majority of points lost on this question were for failing to provide inputs and outputs for the test cases.

This outcome is partially met. The vast improvement would be to get students to recognize that a test case requires inputs and analysis of outputs.

Outcome 2h) Apply static and dynamic techniques to analyze non-functional properties, including common security vulnerabilities such as password weakness, over/underflows, and race conditions.

Assessment This was not covered or assessed.

Analysis

Outcome not met.

Outcome 2i) Engage in self-directed study to learn new techniques and tools for software design, implementation, and/or testing.

Assessment: Project

This outcome is assessed by project success. In the past two semesters, students have developed project using Python, MongoDb, and a variety of other technologies such as Splunk and Radare. It is impossible for teams to succeed without learning about these tools, which they do on their own. Early prototypes of the project include demonstrations of the use of these tools.

Analysis

Assessment of the outcome comes through the students' abilities to download, study, integrate, describe, and demonstrate the use of these technologies in the project.

This outcome is met.

Outcome 3a) Conduct a technical review of software design, implementation, and V&V.

Assessment: Indirect

Through conversations with teams, it is clear that reviews and inspections are occurring; however, it was not measured. Teams members are individually responsible for team submissions, and all teams recognize the need for work reviews. (This is documented in their SCM plans, which universally call for review before submission.)

Analysis

This outcome is not measured and is being assessed informally. I believe it is being met.

Outcome 3b) Create and implement a software configuration management plan.

Assessment: Project deliverable

Teams develop, deliver, and use a configuration management plan. Below is the grading rubric.

Final	**Checklist**
10 pts.	Introduction ❑ Project Description—Does the overview describe the project accurately? ❑ Purpose and Intended Audience: Is the purpose correct and complete? Is the intended audience correct? ❑ References ❑ Overview of Document
25 pts.	Software Configuration Identification Are the SCI at the correct level of granularity? Are all the artifacts that should be put under configuration present? ❑ Naming convention for Items ❑ Directory Structure ❑ Backup plan is complete ❑ Processes to manage above items is described in detail
40 pts.	Software Configuration Control How realistic is the approach? Is it clear how change will be managed? Are there supplemental documents provided?

	❑ Documentation and Process to control changes ❑ Configuration Control Board identified and approval process defined ❑ Detail process description is given on change control ❑ Configuration Manager identified ❑ Baseline creation ❑ Check in/out procedures described
20 pts.	Software Configuration Auditing ❑ Process is described ❑ Responsible parties identified ❑ Related documentation is described
5 pts.	Organization /Grammar /Presentation

Analysis

The grades range from 65 to 93. The class average is 80. Students are questioned about their use of the SCM plan at the end of the semester. Most teams attempt to follow it. Most teams modify the plan during the semester to make it tractable.

This outcome is met.

Outcome 3c) Make use of appropriate architectural styles and diagramming techniques to define an architecture.

Assessment: Team presentations

See discussion of assignment for outcome 1c. All presentations had appropriate modeling diagrams to describe architectures.

This outcome is met.

Outcome 3d) Make use of appropriate textual notations and diagramming techniques to create a detailed design for a software system.

Assessment: Project Deliverable.

Grading Criteria: Completeness, Correctness, and Consistency	**Checklist**
/10 pts.	Overall Structure • Purpose and intended audience • Scope of product • References • Definitions, acronyms, and abbreviations • Overview
/25pts.	Classes, Subsystems, and Components • Correct notation • Classes in subsystems collaborate • UI classes separate from model classes • Database present Design Patterns • Using design patterns correctly • adding the correct classes for the design pattern(s)
/15pts	External Components/Dependencies • Splunk • Graphing tool (Maltego, Graphviz, etc.) • OCR • Speech to Text tool • Mongo, Python, OS, or other dependencies

/35 pts.	Contracts, Responsibilities, and Protocols • Correct notation • Responsibilities collaborate with contracts • Correct representation • Completeness • Complete method signatures • Pre and Post conditions
/10 pts.	Database Schema
/5 pts.	Organization /Grammar /Presentation

Analysis

Grades ranged from 62 to 91. The class average was 73.

SDDs are, in general, weaker than we'd like to see. It's difficult to manage generation of an SDD for the entire project. This might be better served generating the SDD for a subset of the project.

Outcome partially met. Needs improvement.

Outcome 3e) Construct software from a detailed design.

Assessment: Project Deliverable

In 2019 and 2020, we attempted to be more agile/iterative in the development of the team project. The approach was essentially an incremental prototype. Deliverables were due every two weeks.

In this environment, the software was under development while the design was being generated. In most cases, the final product was refactored to be more like the design towards the end of the semester.

Analysis

This outcome should be reconsidered. It is atypical to have a detailed design from which to build software. It is more typical to refactor.

Perhaps some combination of detailed design and iterative design/implementation would be reasonable. For example, do iterative for part of the project and up-front detailed design for another part.

This outcome is partially met, if we accept refactoring as implementing to a design.

Outcome 3f) Develop a test plan for a software system.

Assessment: Project Deliverable

Grading Criteria: Completeness, Correctness, and Consistency	**Checklist**
/10 pts.	Overall Structure • Purpose and intended audience • Scope of product • References • Definitions, acronyms, and abbreviations • Overview
/25pts.	Test Approach • UI Testing Covered • Test Cases mirror use cases • Criticality metric for test cases explained • Test cases that cover: o Ingestion o Graphing

/35pts.	Test Cases • Clear, concise, instructions • Consistent wording for same/similar actions • Specific instructions for testers (I.e. “select file ‘x’”, rather than “select desired file”) • Adequate information to verify test results
/10 pts.	Suspension/Exit Criteria • Reasonable suspension criteria
/10pts	Test Schedule
/5pts.	Other Sections
/5 pts.	Organization /Grammar /Presentation

Analysis

Scores on the test plans ranged from 72 to 98. The average was 82.

Outcome is met.

Outcome 3g) Demonstrate an ability to orally present a software design and implementation.

Assessment Project Presentations

Ordinarily, teams give final presentations and are graded both as a team and individually. In fall 2019, the individual average was 84. In spring 2020, teams put a final presentation together, but were not asked to formally present.

Analysis

This outcome is met.

Outcome 3h) Compose software design-related documents that are grammatically correct and technically sound.

Assessment: Project Deliverables

Project deliverables for 4311 include SCM plan, SDD, test plan, test results, source code, user guide, and an architecture lecture.

Analysis

This outcome is met.

Outcome 3i) Apply effective techniques for collaboration and problem-solving within a team.

Assessment: Ad hoc.

This outcome is not measured. I cover conflict resolution strategies, and work with team on an individual bases as needed or requested. I met with every team at least once during the semester; I met with several teams weekly; the TA met with all teams weekly.

Student Examples: None

Analysis

Team work is challenging; it was considerably more challenging in Spring 2020 once the university closed.

Assessing how effective students are at problem solving is difficult.

While there is evidence that most students are able to function adequately in a team setting, it is not clear how effective this course is in promoting that.

This outcome is not measured.

Level 1: Knowledge and Comprehension	**Comment**
a) Articulate design principles, including cohesion and coupling, encapsulation, and information hiding.	Met
b) Describe software design concerns related to maintenance.	Met
c) Describe different software architectural styles (distributed, cloud, blackboard, event systems, layered system, and pipe and filters).	Met
Level 2: Application and Analysis	
a) Identify appropriate software architectural styles (specifically distributed and cloud) given a software design problem.	Not met. **Suggestion:** Move outcome to level 1. Focus to be on student's ability to identify architectures styles within a system.
b) Apply assertion-based techniques (such as pre- and post-conditions) to specify behavior of program modules.	Partially met. Still a weak point in the course. The move of Discrete Structures to a CS course should improve students' performance in the future.
c) Distinguish between the different levels of cohesion and coupling.	Met
d) Use software development and maintenance tools, such as software documents creation and editing tools, GUI generators, comprehension and analysis tools, supporting activities tools (configuration management tools), verification and validation tools, and security vulnerability analysis tools.	Partially met
e) Describe differences between unit, integration, system, and acceptance testing.	Not assessed. Results of outcome 3f can be used as the assessment of this outcome.
f) Apply black testing techniques to develop test cases for a variety of test coverages.	Partially met
g) Apply white-box testing techniques to develop test cases for a variety of test coverages.	Partially met
h) Apply static and dynamic techniques to analyze non-functional properties, including common security vulnerabilities such as password weakness, over/underflows, and race conditions.	Not met. This should be changed to "not assessed" as the topic(s) was/were not covered in class

i) Engage in self-directed study to learn new techniques and tools for software design, implementation, and/or testing.	Met
Level 3: Synthesis and Evaluation	
a) Conduct a technical review of software design, implementation, and V&V.	Not assessed. Clients' bi-weekly reviews of teams' demos might be used as the assessment instrument here
b) Create and implement a software configuration management plan.	Met
c) Make use of appropriate architectural styles and diagramming techniques to define an architecture.	Met Outcome assessed in terms of students' ability to describe architectures using diagraming techniques and not the ability to apply architecture styles.
d) Make use of appropriate textual notations and diagramming techniques to create a detailed design for a software system.	Partially met **Suggestion:** focus on developing SDD for a partial component of the system instead of the whole system
e) Construct software from a detailed design.	Not assessed. **Suggestion:** It needs to be made clear in the outcome that the SDD must be used to guide the implementation of part(s) of the system and not necessarily the whole system
f) Develop a test plan for a software system.	Met
g) Demonstrate an ability to orally present a software design and implementation.	Met
h) Compose software design-related documents that are grammatically correct and technically sound.	Met
i) Apply effective techniques for collaboration and problem-solving within a team.	Partially met 4311 does not handle this outcome as rigorous as 4310 does. Still there are plenty of opportunities to address the outcome in 4311