Introduction of Software Engineering Concepts for Electrical and Computer Engineering Students and Application to Senior Projects

Dr. Danielle Marie Fredette, Cedarville University

Danielle Fredette received her Ph.D. degree from The Ohio State University's College of Engineering (Columbus, OH) in 2017, her M.S. also from The Ohio State Univeristy in 2016, and her B.S.E.E. from Cedarville University (Cedarville, OH) in 2012, during which time she participated in research as an intern at the Air Force Institute of Technology at Wright Patterson Air Force Base, OH, in the Radar Instrumentation Lab. While researching for her Ph.D, she was a University Fellow and then a GATE Fellow with The Ohio State University's Center for Automotive Research and its Control and Intelligent Transportation Research Lab. She is currently serving as an Assistant Professor of Electrical and Computer Engineering at Cedarville University. Research interests include control for multi-agent systems and autonomous ground vehicles.

Nathan Jessurun, University of Florida

Nathan Jessurun received his B.S. in Computer Engineering from Cedarville University in 2019. Currently he is a PhD candidate at the University of Florida, working toward a degree in Electrical and Computer Engineering. His research interests include x-ray and terahertz wave applications, multi-modal imaging, tomographic reconstruction algorithms, and machine learning applications.

Introduction of Software Engineering Concepts for Electrical and Computer Engineering Students and Application to Senior Projects

Danielle M. Fredette, Nathan T. Jessurun
Cedarville University

Abstract

This paper describes results of incorporating basic software engineering principles into the senior design curriculum for electrical and computer engineering students who have no prior software engineering exposure. Software engineering concepts are introduced to computer and electrical engineering students in the fall of the senior year using lectures, books, and guided application to a year long project. As the electrical and computer engineering fields have broadened, introducing software engineering topics to all of the students before graduation has become increasingly valuable to both students and faculty.

The senior design course itself is described as it is currently along with its evolution over the course of the program's history. Student perspectives are analyzed using comments from the course evaluation as well as taking a closer look at how one project team applied software engineering basics in their project toward greater success and satisfaction in their senior design experience. Several further improvement possibilities for the course are identified from this student feedback, especially regarding the response of the more hardware oriented electrical engineering students to software engineering topics.

1 Introduction

This paper is an experience report on a senior capstone course in which basic software engineering principles are introduced to electrical and computer engineering students who have had no prior exposure. Our intention is to describe a model for the incorporation of introductory software engineering concepts using both classic and newer materials, with a strong emphasis on project based application of the concepts in the senior capstone design.

In this paper, we report on the evolution and evaluation of our school's senior design capstone class for electrical and computer engineering (EE/CpE) students. Specfically, we will look at the impact of the inclusion of introductory software engineering materials on senior design performance for students with little to no knowledge about large software project management. We will begin by describing and discussing changes made to the content/seminar portion of the course. Then we will discuss student feedback on the course from student evaluations and take a

closer look at one team's particular experience. The purpose of this paper is to communicate what has and has not worked well for us in the senior design course over the years, and to draw some preliminary conclusions about how EE/CpE students respond to course content on the basics of software engineering, and how they do or do not make use of that material in their projects, with the intent to improve our course for future years. Specific items to evaluate over the next cycle of the course are also identified.

As noted by the author of [1], senior capstone design work is a valuable part of an engineering curriculum, giving many students their first experience working on a team to complete a large project on a relatively long time scale. However, many students, especially those with less technical experience from internships or co-ops, come into senior design with weaknesses. These weaknesses might include unfamiliarity with how to plan for and manage constraints such as time and money, discomfort with being dependent on teammates, lack of experience regarding how to manage a team or a meeting, and unreasonable expectations about system integration. In our school's EE/CpE senior design program, we attempt to head off some of these weaknesses by offering a seminar/lecture component in the fall part of the course in which we teach some of these soft skills (similar to what is described as typical in [2]). Software engineering principles, along with instruction on more general project management and leadership, form the basis of the seminar content. Throughout the year, each team is guided by a faculty advisor toward putting these ideas into practice on a variety of hardware and software projects.

2 Capstone Course Development

2.1 Senior Design Course Overview

Cedarville University's senior design or capstone class is a year long project-based course. Teams of 2-6 students work together for the entire school year on a large project under the supervision of a faculty advisor. Each team's project is typically unique from that of other teams, and both students and faculty are involved in proposing and selecting projects and teams for the upcoming school year. Teams are typically multi-disciplinary with a mixture of electrical and computer engineering students. Using the classification criteria of [2], our senior design class is 1) offered at the program level and 2) gives students authentic involvement in solving a real problem. Some projects have real customers (business, nonprofit organization, other university department, engineering competition team). Almost every project has both hardware and software components. Students are allowed to self-select their teams, with certain project and team size constraints given by the faculty.

During the fall semester, in addition to the project-based part of the class, students participate in a seminar-type course. The course has three major facets: 1) to teach the principles of project and team management, 2) to teach software engineering principles and practices, and 3) to lay the foundation for the spring by developing project proposals and beginning a design cycle for a product. The objectives of the fall senior design course, mapped to ABET outcomes, are the following:

1. Identify and apply principles of project management. (2, 5)
2. Identify and apply key concepts of electronic product design.(1, 2)

3. Demonstrate effective project communication skills.(3)
4. Work with others in solving a realistic design project.(5)
5. Select and use computer aids appropriate to various aspects of electronic design. (1,7)

Grades for the course are awarded based on the following breakdown.

- mini projects: 12%
- reading/class presentations: 15%
- project proposals: 15%
- project milestones (2): 10%
- weekly reports: 5%
- hours logs: 5%
- final presentation/design report: 15%
- instructor evaluation: 23%

The schedule of topics and assignments for the course is shown in Figure 1. Note that, for the latter half of the course, the students are asked to give presentations on the book material. This approach has been adopted both to help the students truly engage with at least a subset of the required reading material and to give all of them additional practice presenting to a group.

Students typically find senior design to be a rewarding opportunity. They enjoy being able to create something they can be proud of, work closely with a faculty member and with other students on a realistic engineering project, and gain valuable hands-on experience.

EGEE/EGCP-4810 Class Schedule 2019					
Date	**Reading**	**Speaker**	**TOPIC**	**Thurs Lab**	**Assignments Due**
Wed, Aug 21	Design CH 1	Professor	Intro: The Engineering Design Process		
Fri, Aug 23	Design CH 2	Professor	Project Selection & Needs Identificcation	Team meeting	
Mon, Aug 26	Design CH 3	Professor	Requirements Specification		
Wed, Aug 28		Professor	Writing your proposals and requirements	Work time, PCB meeting	officer designations
Fri, Aug 30			Team Meeting		
Mon, Sep 2	Labor Day		*Labor Day - No class*		
Wed, Sep 4		Professor	Technical Writing	Work time. PCB meeting	
Fri, Sep 6			Team Meeting		
Mon, Sep 9	Design CH 9,10	Professor	Team & Project Management		
Wed, Sep 11	Design CH 5	Professor	Functional Deomposition		
Fri, Sep 13			Team Meeting	Work time	
Mon, Sep 16	Design CH 7	Professor	Testing		
Wed, Sep 18	Design CH 8	Professor	Reliability		
Fri, Sep 20			Team Meeting	Work time	formal proposals
Mon, Sep 23	Myth CH 1-3	Professor	Tar Pit, Mythical Man-Month, Surgical Team	Work time, machine learning meeting	
Wed, Sep 25	Career Fair		Required attendance at the Career Fair		
Fri, Sep 27			Team Meeting		mini-project #1
Mon, Sep 30	Leadership 1&2	Professor	Greatest is the Servant/Show Justice Mercy		
Wed, Oct 2	Agile CH 1-2	Professor	Development & Begin Agility	Work time, CAN meeting	
Fri, Oct 4			Team Meeting		
Mon, Oct 7	Myth 4-5	ML game	Mythical Man Month #2	Design Reviews: milestone #1 (your team advisor)	
Wed, Oct 9	Leadership 3&4	ML camera	Seek to Serve / Share Credit, Shoulder Blame		
Fri, Oct 11			Team Meeting		mini-project #2
Mon, Oct 14	Agile CH 3	Car	Feeding Agility		
Wed, Oct 16	Myth 6-7	Aero	Mythical Man Month #3		
Fri, Oct 18			*Fall Break - No Class*	*Fall Break*	
Mon, Oct 21	Agile CH 4a	Arm	Delivering what users want (10-14)		
Wed, Oct 23	Leadership 5&6	Radio	Ask for Help / Keep a Short List		
Fri, Oct 25			Team Meeting	Work time	mini-project #3
Mon, Oct 28	Agile CH 4b	Boat	Delivering what users want (15-18)		
Wed, Oct 30	Leadership 7&8	ML game	Read to Lead / Set the Course … and Pace		
Fri, Nov 1			Team Meeting	Work time	
Mon, Nov 4	Agile CH 5	Car	Agile Feedback		
Wed, Nov 6	Leadership 9-11	Car	Accountablility /Exercise/ Conclusion		
Fri, Nov 8			Team Meeting	Work time	mini-project #4
Mon, Nov 11	Agile CH 6	Aero	Agile - coding		
Wed, Nov 13	Agile CH 7 & 8	Radio	Agile Debugging & Collaboration		
Fri, Nov 15			Team Meeting	Work time	
Mon, Nov 18			Potential Make-Up Day	Design Reviews: milestone #2 (your team advisor)	
Mon, Nov 18					
Wed, Nov 20					
Fri, Nov 22					
Wed, Nov 20	No Class		*Thanksgiving Break*		
Fri, Nov 22	No Class		*Thanksgiving Break*	*Thanksgiving*	
Mon, Nov 25	No Class		*Thanksgiving Break*		
Mon, Nov 25					
Wed, Nov 27				Presentations	
Fri, Dec 6					
Sun, Dec 8			Design Reports Due by 12:00 noon	1st PCB Design (if you have a PCB)	final design reports

Figure 1: Fall 2019 Senior Design Course Schedule

2.2 Improvements Through the Years and Software Engineering Content

The engineering program at Cedarville University began in 1990, initially offering majors in mechanical and electrical engineering. The first electrical engineering graduating class was the class of 1994, and initial accreditation was received that same year. The computer science (CS) program graduated its first students in 2002, and its accreditation came in 2006 after moving from the department of science and mathematics to the department of engineering and computer science. The computer engineering program was conceived as a hybrid of electrical engineering and computer science, and was first offered in the year 2002, with its first graduating class and accreditation in 2006.

A senior design capstone project has always been part of the curriculum for engineering students, with electrical engineering and computer engineering students put together on teams in order to utilize their different skills to complete a realistic project. Initially, all of the EE student teams did the same project for a full year, with each year's project being different from the previous year's. After about 5 years of this, senior design started having diverse projects. Each project and team is advised by a faculty member, and each year both students and faculty get an opportunity to propose new projects.

In addition to the year-long project-based objectives of the senior design course, EE, CpE, and CS students all receive a senior design lecture component in the senior fall semester. The goal has always been to introduce useful topics (such as software engineering) to the students in a way that contributed to their senior design projects and future work aspirations without taking too much time away from the project component of the course. The EE content has included project management concepts plus certain technical topics thought to aid the students in their projects (power supplies, voltage regulators, communication protocols, etc.). In the course of ongoing accreditation cycles and recommended improvements to the curriculum, it was decided that CpE students should receive some formal instruction on the ideas of software engineering. This was accomplished initially by requiring CpE seniors to join the fall lecture portion of the CS senior design course. This did expose CpE seniors to useful software engineering concepts, but it caused them to feel a bit "orphaned" or caught between two programs, being in lecture with CS seniors but on project teams with EE seniors. It was also difficult to keep senior design course grading and expectations well-communicated and consistent. For these reasons and because the CS program experienced significant growth, a CpE faculty member took over the CpE senior design course in 2014. Under this scheme, the CpE students shared a few lectures on project management with the EE class, and the rest of the course content was borrowed directly from the CS senior design curriculum.

Between the years of 2014 and 2018, the CpE senior design curriculum underwent considerable change. Initially, Pressman's traditional textbook *Software Engineering: A Practitioner's Approach* [3] was used, giving way to the more concise Gustafson's *Schaum's Outlines: Software Engineering* [4]. Neither of these seemed the right fit for the CpE students, and eventually the CpE faculty settled on covering the material found in the following collection of shorter books, which comprise the current text choices for CpE senior design:

- *Design for Electrical and Computer Engineers,* Ford and Coulston [5]
  - Project management and teamwork basics
  - How to organize, decompose, and plan a large project
  - Defining requirements, objectives, needs, constraints, standards
- *10 Leadership Maneuvers,* Loren Reno [6]
  - Wisdom on how to be a good leader
- *The Mythical Man-Month, Essays on Software Engineering,* Frederick P. Brooks Jr. [7]
  - Classic but relevant wisdom of a father of the software engineering field
  - Explores philosophy and experience surrounding large software projects and their unique difficulties
- *Practices of an Agile Developer,* Venkat Subramanaim and Andy Hunt [8]
  - An introduction to the Agile methodology
  - Do's and don'ts of practical team software projects

In 2018, the faculty experienced some turnover and subsequent shuffling of loading. At this point it became clear that the EE senior design lecture curriculum also could use an update. It had previously covered many useful, miscellaneous technical topics, but with the growth of the fields of electrical and computer engineering and the blurring of the lines between them over recent decades, the faculty decided that the EE students would also benefit from an introduction to software engineering and merged the EE and CpE senior design courses entirely starting in 2019. The most important technical topics from the two sections were converted into a bank of assignments from which the students could choose the four that most interested them.

Next, we will look at examples of how one team utilized the software engineering principles learned in class toward a successful senior design project.

3 Case Study: Autonomous Shuttle Project

3.1 Autonomous Shuttle Project Overview

Beginning in fall 2018, the authors were part of the team working on a new autonomous shuttle senior design project. D. Fredette served as the faculty advisor and N. Jessurun was one of four team members. The goal was to convert a small ground vehicle, such as a golf cart, into a fully autonomous shuttle for urban/on campus transport, with similar functionality to existing experimental autonomous shuttles [9]. Part of the challenge was a minimal budget of just a few thousand dollars. Figure 2 depicts the used golf cart our team acquired for the project and a student using a GPS receiver to collect waypoints for navigational use.

The year 1 team accomplished converting the golf cart to a drive by wire vehicle (designing and building systems for electronic steering, braking, and throttle, Figures 3 and 4) and demonstrating autonomous navigation by following GPS waypoints. The Reach GPS unit (Figure 5) provided

sufficient accuracy to stay on a sidewalk when used with real time kinematic (RTK) correction from a nearby Ohio Department of Tranportation GPS base station. Software and algorithms for obstacle avoidance were also developed, although the full implementation of an obstacle avoidance system was left to a future team. Year 1 managed to realize level 3 autonomy (steering and brake/acceleration capability of the car such that the computer can take over driving when certain conditions are met) [10] with a budget of about $3500.

Regarding software design, the students looked carefully at similar applications [11]–[17] to identify common elements for a robust software architecture for autonomous navigation. We identified for such elements. They are listed below with a brief description of how each was implemented in the design.

- Acquisition of navigational data
  - Waypoints and position/speed information from EMLID Reach+ GPS unit, Figure 5
  - Obstacle detection using RADAR and/or LIDAR
- Asynchronous, separate processing routines for unique autonomous operations
  - Software must be able to continue processing messages from subsystems with differing or inconsistent update rates. For example, if the GPS sensor momentarily loses satellite connection, the system should still be able to process incoming information from other sensors.
- Robust communication interfaces to facilitate data transfer between sensing components
  - Figure 6 depicts the software architecture of the internal communication system we implemented. To ensure consistency between modules, we programmed a NodeInterface, the concept of which is depicted in Figure 7.
- Priority-based feedback from the processor to each locomotive component (e.g. motors controlling vehicular movement).
  - Priority information is built in to the ConnectionHub design. That is, each update from a module has a priority attached, and updates are dispatched in priority order.

The autonomous car project is now in its second year, with a new team of six seniors adding LIDAR functionality, dynamic routing algorithms, and more robust hardware improvements. Starting drive by wire conversion on a second golf cart is also in the works, which will allow us try out some multi-vehicle interactive driving possibilities, perhaps in year three.

Figure 2: Photo of the golf cart during initial waypoint collection

Figure 3: The golf cart after drive by wire conversion

3.2 Student Perspective: Lessons Learned from Applying Software Engineering Basic Principles to the Senior Design Project

As a recent graduate, I (N. Jessurun) have found Cedarville University's senior design to be a highly effective course which prepares its graduates both for industry and advanced academics. Students are taught teamwork, project management, and interdisciplinary skills – increasing their technical and social aptitude in the process. In this section, I will discuss some of the ways the autonomous car team and I were able to apply and test the software engineering principles we learned in class.

Part of the senior design class involved reading a book from the developers of Agile principles [8] and incorporating these practices into our design process. This entailed breaking deliverable products into 1- to 2-week cycles, with definite goals reached each iterative cycle. Agile requires teams to break large tasks, such as designing a system to act upon GPS data, into small, bite-sized pieces. In the GPS example, this would involve turning the overall goal into objectives able to be completed in 1-2 weeks like:

- Design an interface to poll the EMLID system for positional data

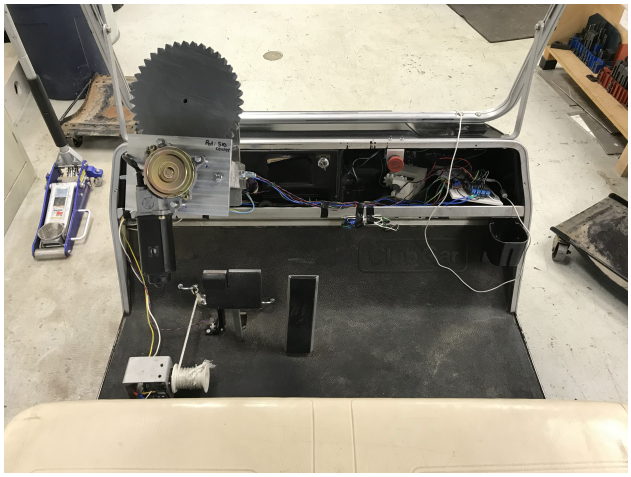

Figure 4: Drive by wire hardware for braking and steering

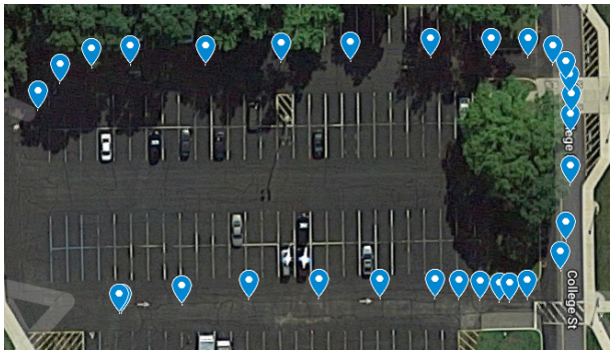

Figure 5: The Reach GPS unit and an example of the waypoints collected along a driving path. Data plotted using `maps.google.com`.

- Develop an algorithm to extract heading and velocity data from progressive geographical timestamps
- etc.

The Agile mindset proved highly useful to our senior design team. Using Trello, an online list-based task scheduler, we were able to record time spent on each task and ensure each team member received an equitable work load. With this system in place, milestones felt far more achievable.

Our team performed a small 'case study' of Agile-based vs. traditional waterfall release tasks. At certain times throughout the semester, multiple courses have projects or exams at the same time, requiring more work than usual for students to keep up. During one such period, our senior design team didn't take the time to break larger milestones into smaller tasks. Instead, we each logged time to a single, high level descriptor for the milestone, such as 'Complete GPS program'. Uncharacteristic of previous accomplishments, we were unable to meet our specified goals and didn't implement the desired level of functionality. We had some doubts as to whether the lack of smaller tasks and realistic goals was to blame (instead of just a lack of time). However, an

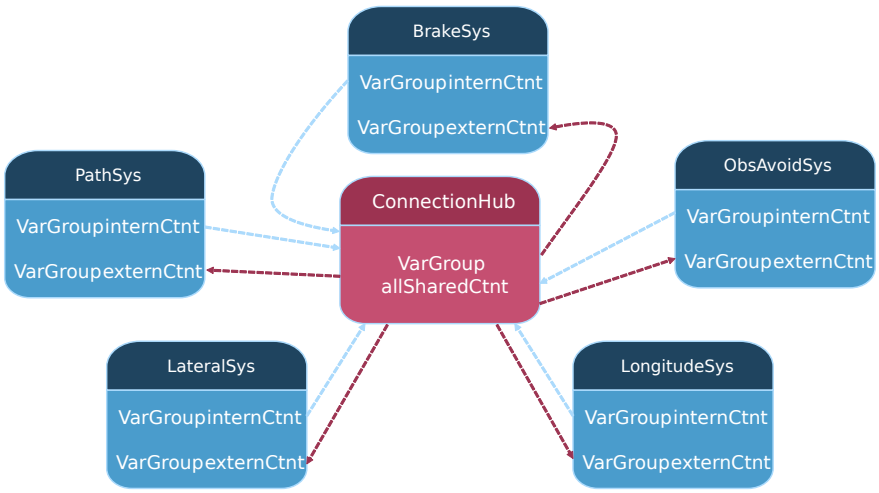

Figure 6: Diagram of the connection hub software architecture for overall data sharing between software modules in the project

opportunity to test this hypothesis presented itself when in the next semester a similar priority conflict arose. This time, however, the team lead dedicated time to appropriately break apart tasking between members. The additional planning paid off – time logs show we avoided spikes in hours recorded, which shows we properly estimated our capabilities. When bite-sized objectives were consistently defined for each overarching responsibility, the team was much more effective at realizing them. At the end of the year, each member noted in retrospect the additional work required to break down large milestones proved worthwhile.

Agile methodology is helpful for naturally and realistically breaking large objectives into smaller tasks. However, it lacks the ability to *generate* initial high-level goals from a single product statement. The textbook *Design for Electrical and Computer Engineers* [5] was included in the senior design curriculum to address these concerns.

While *Design for Electrical and Computer Engineers* contains several helpful principles for system design, we found requirement specifications to be the most beneficial and applicable to our project. Namely, we used these guidelines to break our ambiguous product description into several separate, *testable* objectives.

One recommendation given by the book was to avoid thinking about a project in terms of what must be designed. Instead, the author suggested focusing on verifiable actions – termed 'product requirements' – that the system can perform. Next, these requirements are grouped by items with related functionality. Breaking our project description down this way, we were easily able to identify the high-level goals of our project. For instance, our decomposition of navigation requirements resembled the following, starting from the project objective and growing more specific as you go down in hierarchy:

1. The team shall design and demonstrate capabilities of an autonomous vehicle.
    (a) The cart shall accelerate when given the appropriate software command
        (i) On human intervention (i.e. a keypress), all autonomous acceleration instructions

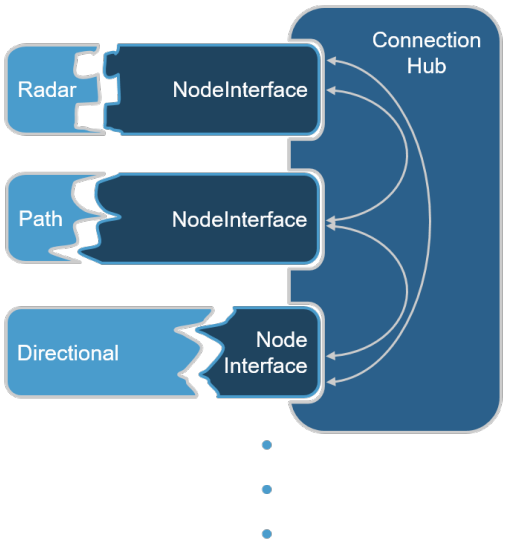

Figure 7: Connection Hub software architecture concept

shall cease.

(ii) The commanded acceleration shall not exceed appropriate speed limits for the areas traversed

(iii) ...

(b) The cart shall slow when given the appropriate software command

(c) The cart shall turn when given the appropriate software command

(d) ...

(e) The cart shall navigate routes of reasonable complexity without requiring human intervention.

Once enough points are fleshed out, similar requirements can be grouped and separate systems become more recognizable. Hence, top-level system designations can be generated by thinking about what *operations* are required, not what structures must be in place to accomplish them. More concisely, this entails creating an architecture first answering *what* the system does, and only then considering *how* this is realized.

Beyond identifying objectives, product requirements also ensure that no portions of the system are incompatible with each other. While determining software goals, we realized that two system components performed mutually exclusive functionality. This was indicative of a poor design, which we were able to rectify before incurring significant time losses.

In our senior design course, we were also required to read and discuss the book *The Mythical Man Month* [7], a collection of Fred Brooks' classic essays on software engineering. This book offers insightful comments on the history of large software design. Since we were creating our autonomous vehicle's navigational framework from scratch, it was helpful to keep these ideas in

mind as we proceeded with our design. In particular, Brooks' essays provided stories contrasting newer, experimental solutions to mature alternatives – the same situation we were in during one stage of software development.

After splitting software functionality into several separate modules, we needed to create a framework in which these pieces could share information with each other. This is a problem encountered in a multitude of software projects, so multiple existing libraries provided various solutions. Using Brooks' insight, we opted for a solution leveraging the appropriate balance between stability and performance. This lead us to select a JSON serialization library from an established company, indicating it would receive sustained development. Moreover, this library was created from scratch in 2011 to provide a lightweight, fast alternative to standard JSON encoding modules.

The Mythical Man Month also discusses the pros and cons of efficient task reporting. Referring to some of the author's stories, we attempted to find the right balance between easy reporting and avoiding unnecessary administrative overhead. Our team found that Trello fit nicely between these two bounds. Since each task was manually created and assigned a due date, everyone was aware of each current objective. However, the automated notification systems and time tracking capabilities kept all of us up-to-date without a large amount of back-and-forth communication.

4 Student Response to Fall Course Content: Course Evaluations

While the previous section represented the experience of one senior design team, in this section we will take a look at a broader sampling of student opinions on the basic software engineering content from representative course evaluation comments. These comments came at the end of the fall semester of 2019.

Most negative comments reflect the students' desire to spend less time in lecture in general and more time on their projects, as exemplified by the following two comments copied from the course evaluation.

> *I extremely disliked the lectures and the other assignments and readings for this class. They were pointless and a waste of my time. I just wanted to work on my senior design project and the other assignments and readings for this course took up time I could have spent on my project.*
>
> *I thought the first portion of book club was helpful, but the latter seemed a little unnecessary. Rather than having a bunch of presentations, I think it may have been more beneficial to have each person write up a < 1 page summary/report on various sections of the books, and submit that. The time spent in class for the presentations would then be able to be spent on our actual senior design project.*

Other students, however, enjoyed reading and discussing the books together as a class and had at least a subset of the books that they found to be useful:

> *I really appreciated taking the time to go through the books, particularly the leadership one!*

> *I'm very grateful that the faculty dropped the tests in favor of the mini projects. It allowed us to focus more o the practical side of the project. Additionally, I really enjoyed the book club. I wasn't a huge fan of the first book we went through in the course, but I really enjoyed going through the agile book, leadership book, and mythical man month.*

The Agile book [8] was and continues to be immensely popular with the CpE students. Many are quick to pick up its recommendations with respect to how they run their project teams. Agile is, as is known, a prominent tool for software engineers in both industry and academia, and many colleges, ours included, have incorporated it into their CS and CpE curricula for years [18]–[21]. 2019, however, was the first year at our school that it was taught to EE students as well, in hopes that project teams will be more unified in their knowledge and therefore more likely to try the structures and ideas out for themselves.

While our EE students are increasingly finding it necessary to engage in at least some level of programming in their projects and internships, some of them are less interested in software than others. Hardware people tended to appreciate the Ford and Coulston book more, while programmers enjoyed the book on Agile. Some EE students thought the software engineering content specifically to be irrelevant to them and their projects, as exemplified by the following couse evaluation comment.

> *The course assigned multiple books to read for class. Two of the books seemed to be more applicable to Computer Engineers and not Electrical Engineers. This was fine, from an EE perspective, but I did feel like many lectures did not apply to me.*

In the discussion section, we will further address concerns brought up in the student evaluations.

5 Discussion

5.1 Future Course Improvements for Senior Capstone

Regarding student conerns, I (D. Fredette) first noted that the students appreciated the streamlining of the course that happened for the 2019 year and the consistency between the two majors. This was not represented in the course evaluation comments from 2019 because not all students knew about the way the course had been run in the past. However, conversations with students from both classes (2019 and 2020) suggest that keeping the expectations the same for both EE's and CpE's is an improvement since EE's and CpE's are put on teams together.

The main student complaint is that they would prefer there to be no or a significantly reduced lecture component to the fall course (there already is no lecture component to the spring course). This concern had already been addressed significantly between the 2018 and 2019 years by reducing the lectures from three times to twice weekly, freeing up an hour for a weekly team meeting, and ending them by Thanksgiving break. The 2019 class also was allowed to choose their own subset of previously offered technical content from a bank of assignments and special seminars, freeing up more time while still providing instruction for those who need it on topics students deem relevant to their interests. The faculty agree that some lecture/technical content is

helpful, but I believe that more careful curating is called for. For example, I would like to reduce the number of chapters covered from the Ford and Coulston book, focusing more on the requirements specification and cutting the less relevant/engaging material.

One important student concern comes from the EE students who do not believe software engineering basics to be applicable to them. It is true that each year we will have one or two projects that are almost entirely hardware oriented. For these students, and because all of them need to be prepared to work on projects that will be a mixture of hardware and software, I plan to add a lecture/discussion on how the Agile methodology can and has been adapted for hardware projects [22], [23], perhaps assigning some additional reading on the topic. We can spend some time discussing the differences and similarities between hardware and software projects and how a hardware subteam can best interface with a software subteam while working under an Agile-type planning structure.

Although not all students immediately see the value in official class discussions on software engineering, those who have experience or get experience by trying out the ideas have found software engineering instruction to be an important part of the course. A greater awareness of software engineering principles among the faculty advisors (most are from an EE background) would be of help in encouraging students to give the practices they learn about in class a try in their own team settings, allowing deeper, experiential learning to take place such as what N. Jessurun described in section 3.2.

5.2 Possible Future Work

In this paper, we have been able to look at a limited set of students and their reactions to exposure to software engineering content in the electrical/computer engineering senior desing capstone. While some relevant comments from course evaluations and a firsthand account were available as evidence, much of the idea of students' valuations of the course content come from unofficial feedback or instructor experience. It may be of interest to conduct more targeted surveys of specifically electrical engineering students to understand their impression of the material and ideas about its usefulness to them, perhaps both before and after the course. An orderly evaluation of this sort is left as future work.

6 Conclusion

Senior design has been an important part of the engineering program at Cedarville University since the program's founding 30 years ago. This paper outlined the evolution of the senior design curriculum for electrical and computer engineering students over that time. Classroom discussions on software engineering topics have been an important part of the computer engineering curriculum, but only recently have been included for electrical engineering students as well. The current senior design model is fully interdisciplinary including electrical and computer engineering students doing the same coursework and populating project teams together. The students and faculty enjoy the consistency and, as the lines between the expectations on electrical and computer engineers continue to blur, we believe that everyone benefits from learning some basic software engineering ideas.

In this paper we also described how one senior design team was able to test out some of the software engineering and design principles learned in class, leading to greater project success and satisfaction than would likely have been possible otherwise. While student evaluation responses on the course were mixed, they point to several specific measures that can be enacted to improve the course next year. Specifically, improvements include further curating and prioritization of existing content so that students feel that their time is well spent, and a pointed discussion on how a version of the Agile methodologies can and have been used for hardware development as well as software. When students are equipped to put what they learn into practice with hands-on projects, they are able to experience and remember the content in a way that they can carry with them into their careers. We hope to encourage more of this in the future for all of our students.

References

[1] J. R. Goldberg, "Preparing students for capstone design [senior design]", *IEEE Engineering in Medicine and Biology Magazine*, vol. 28, no. 6, pp. 98–100, 2009.

[2] A. J. Dutson, R. H. Todd, S. P. Magleby, and C. D. Sorensen, "A review of literature on teaching engineering design through project-oriented capstone courses", *Journal of Engineering Education*, vol. 86, no. 1, pp. 17–28, 1997.

[3] R. S. Pressman, *Software engineering: a practitioner's approach*. Palgrave Macmillan, 2005.

[4] D. Gustafson, *Schaum's Outline of Software Engineering*. McGraw-Hill, Inc., 2002.

[5] R. Ford and C. Coulston, *Design for Electrical and Computer Engineers*. McGraw-Hill, Inc., 2007.

[6] L. M. Reno, *10 Leadership Maneuvers: A General's Guide to Serving and Leading*. Deep River Books, 2015.

[7] F. P. Brooks Jr, *The mythical man-month (anniversary ed.)* Addison-Wesley Longman Publishing Co., Inc., 1995.

[8] V. Subramaniam and A. Hunt, *Practices of an agile developer: Working in the real world*. Pragmatic Bookshelf, 2006.

[9] DriveOhio, *New self-driving shuttle rolls around scioto mile in columbus*, `http://drive.ohio.gov/news/New-Self-Driving-Shuttle-Rolls-Around-Scioto-Mile-in-Columbus/index.html`, cited September 2019.

[10] "Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles", SAE International, Standard, Jun. 2018.

[11] A. Canedo-Rodriguez, V. Alvarez-Santos, C. V. Regueiro, R. Iglesias, S. Barro, and J. Presedo, "Particle filter robot localisation through robust fusion of laser, wifi, compass, and a network of external cameras", *Information Fusion*, vol. 27, pp. 170–188, 2016, ISSN: 1566-2535. DOI: `10.1016/j.inffus.2015.03.006`.

[12] A. Finzi, F. Ingrand, and A. Orlandini, "Planning and robotics (planrob-15)", p. 148,

[13] S. Hernandez and F. Herrero, *Autonomous navigation framework for a car-like robot*. 2015.

[14] S.-H. Joo, S. Manzoor, Y. G. Rocha, H.-U. Lee, and T.-Y. Kuc, “A realtime autonomous robot navigation framework for human like high-level interaction and task planning in global dynamic environment”, p. 4,

[15] I.-S. Kweon, Y. Goto, K. Matsuzaki, and T. Obatake, “CMU sidewalk navigation system: A blackboard-based outdoor navigation system using sensor fusion with colored-range images”, in *Fall Joint Computer Conference*, IEEE, 1986.

[16] M. Muñoz–Bañón, I. del Pino, F. A. Candelas, and F. Torres, “Framework for fast experimental testing of autonomous navigation algorithms”, *Applied Sciences*, vol. 9, no. 10, p. 1997, Jan. 2019. DOI: 10.3390/app9101997.

[17] J. L. Sanchez-Lopez, J. Pestana, P. de la Puente, and P. Campoy, “A reliable open-source system architecture for the fast designing and prototyping of autonomous multi-uav systems: Simulation and experimentation”, *Journal of Intelligent and Robotic Systems*, vol. 84, no. 1, pp. 779–797, Dec. 2016, ISSN: 1573-0409. DOI: 10.1007/s10846-015-0288-x.

[18] B. Lu and T. DeClue, “Teaching agile methodology in a software engineering capstone course”, *Journal of Computing Sciences in Colleges*, vol. 26, no. 5, pp. 293–299, 2011.

[19] T. Reichlmayr, “The agile approach in an undergraduate software engineering course project”, in *33rd Annual Frontiers in Education, 2003. FIE 2003.*, IEEE, vol. 3, 2003, S2C–13.

[20] S. Cleland and S. Mann, “Agility in the classroom: Using agile development methods to foster team work and adaptability amongst undergraduate programmers”, *16th Annual NACCQ*, 2003.

[21] D. F. Rico and H. H. Sayani, “Use of agile methods in software engineering education”, in *2009 Agile Conference*, IEEE, 2009, pp. 174–179.

[22] K. Thompson, “Eleven lessons learned about agile hardware development”, [Online]. Available: https://www.agileforhardware.com/whitepapers.

[23] ——, “Agile processes for hardware development”, 2015. [Online]. Available: https://www.agileforhardware.com/whitepapers.